UNIFRAME MOBILE AGENT BASED

RESOURCE DISCOVERY SERVICE (MURDS)

TR-CIS-1122-03

Jayasree Gandhamaneni   June 28, 2004

		Form Approved
Report Documentation Page		OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **JUN 2004**	2. REPORT TYPE	3. DATES COVERED **00-00-2004 to 00-00-2004**
4. TITLE AND SUBTITLE **The Uniframe Mobile Agent Based Resource Discovery Service**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Indiana University/Purdue University,Department of Computer and Information Sciences,Indianapolis,IN,46202**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**
13. SUPPLEMENTARY NOTES
14. ABSTRACT **see report**
15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT **Same as Report (SAR)**	18. NUMBER OF PAGES **251**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

THE UNIFRAME MOBILE AGENT BASED RESOURCE DISCOVERY SERVICE

A Technical Report

Report Number

TR-CIS-1122-03

Submitted to the Faculty

Of Purdue University

By

Jayasree Gandhamaneni

In Partial Fulfillment of the

Requirements for the Degree

Of

Master of Science

June 2004

To my Parents, Grand Mother, Deepak and Aryan

ACKNOWLEDGMENTS

As a student at the Department of Computer and Information Sciences, Indiana University-Purdue University-Indianapolis, it has been a wonderfully gratifying experience realizing the dream of studying Masters in Computer Sciences. I am positive that the experience and knowledge I gained here will have a tremendous impact on my life and career.

I would like to take this opportunity to thank many people who have assisted me in successfully completing this project.

First of all, I would like to thank Professor Rajeev Raje, my advisor, for his constant support and able guidance through out the course of my graduate studies and project work.

My special thanks to Professor Andrew Olson and Professor Jeffrey Huang for being on my advisory committee and providing proper guidance during important periods of the project work.

Also, I would like to take this opportunity to thank all my friends who rendered their help in completing this project work when required. I would like to express my great regards for them.

Many thanks to the faculty, staff and colleagues at the Department of Computer and Information Science for their timely assistance and co-operation.

Finally, I would have to thank my wonderful parents and beloved husband Deepak for their constant encouragement and support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Gandhamaneni, Jayasree, M.S., Purdue University, June, 2004. "UniFrame Mobile Agent based Resource Discovery Service". Major Professor: Rajeev Raje

The development of a Distributed Computing System (DCS) using geographically scattered heterogeneous software components is a growing trend. The UniFrame paradigm provides an approach for automatic or semi-automatic creation of a DCS by seamless integration of software components taking into account both functional and non-functional (such as QoS) requirements. UniFrame uses the UniFrame Resource Discovery Service (URDS) for dynamic discovery and selection of components that are deployed on the network. The entities involved in the URDS uses request-reply based communication to accomplish their tasks. These discovery entities periodically discover newly registered components from the native registries. They require certain amount of network resources to effectively discover components. Also, the native registries may decide to offer differentiated services to different discovery entities based on pre-determined policies. The UniFrame Mobile Agent based Resource Discovery Service (MURDS) project addresses the issues related to network resource consumption and heterogeneous policies by replacing the request-reply based communication in the URDS with the mobile agent-based communication. A prototype is designed and experimented with to validate the inclusion of mobile agents in the URDS architecture. The results obtained indicate that the mobile agent-based communication is comprehensive enough to reduce network resource consumption and to interoperate with the native registries that offer differentiated services to different discovery entities.

1. INTRODUCTION

Development of component-based software applications using Commercial off the shelf (COTS) software components is becoming a trend in the field of Distributed Computing Systems (DCS). Some of the reasons for this trend could be attributed to decreased development time and the cost of an application due to readily available components, and increased reliability of systems with the usage of well built and well tested components. However, many issues arise during the development of component-based solutions to DCS. One of these issues is that the components used in the development of a particular DCS may follow different component models such as Java Remote Method Invocation (RMI) [1], Distributed Component Object Model (DCOM) [2], Common Object Request Broker Architecture (CORBA) [3], and .NET [4]. Most of these models provide interoperability in terms of underlying hardware, and operating systems, but do not provide interoperability with components that belong to different distributed component models because of differences in the implementation language or the underlying object model. Hence, an approach is required for seamless interoperation of components developed under different models.

Another issue that needs to be addressed is the Quality of Service (QoS) of individual components as well as DCS developed from those components. ISO [5] defines QoS as "The totality of features and characteristics of a product or a service that bare on its ability to satisfy stated or implied needs". This implies that the QoS assurance of individual components plays a critical role in developing DCS with predictable quality. Composing systems out of COTS poses the need for a standardized mechanism that not only guarantees the QoS of each component, but also of DCS composed out of them. Hence, a framework is needed that incorporates QoS as an inherent part of software components and provides for quantification, verification, validation and specification of both software components and DCS built out of them.

The Unified Meta-component Model Framework (UniFrame) [6, 7] provides an approach for a seamless integration of heterogeneous software components taking into account both functional and non-functional (such as QoS) requirements to build a DCS.

The key concepts of this framework are as follows: a) a meta-component model (the Unified Meta Model – UMM [8]), with a hierarchical setup for indicating the contracts and constraints of the components and an associated discovery mechanism for locating components used in the creation of a distributed system, b) an integration of the QoS at the individual component and distributed application levels, c) the validation and assurance of the QoS, based on the concept of event grammars, and d) generative rules, along with their formal specifications, for assembling an ensemble of components out of available component choices. This project focuses on providing an architecture and implementation, based on mobile agents, for the discovery aspect of UniFrame.

1.1 Problem Definition and Motivation

With the advent of the Internet and its related technologies, a new trend has started towards publishing service provider components on the Internet and the subsequent discovery of these components using directory-based discovery services to build distributed computing applications. Several directory-based discovery services, such as Universal Description, Discovery and Integration (UDDI) [10] registry, CORBA Trader Services [11], Lightweight Directory Access Protocol (LDAP) [12], Domain Name Service (DNS) [14], JINI [25], Service Location Protocol (SLP) [26,27], Ninja Project [28,29], and Universal Plug and Play (UpnP) [32,33] are available to discover service provider components on the Internet. Most of these discovery services use a publish-subscribe model where service providers register their services with a central directory and service consumers discover services by querying respective directories. However, most of these discovery services assume the presence of a homogeneous environment in terms of the component model used to develop the components to be discovered. In a realistic scenario, such might not be the case. Hence, there is a need of a discovery service that uses a publish-subscribe model and at the same time addresses heterogeneity by enabling discovery of components belonging to different existing distributed component models.

UniFrame Resource Discovery Service (URDS), defined under Unified Meta-Component Model (UMM) [8], provides the necessary infrastructure for dynamic

discovery of heterogeneous software components that offer and utilize services as well as the selection of components meeting the necessary functional and non-functional requirements (such as desired levels of Quality of Service) [9]. The URDS infrastructure consists of a component broker, zero or more service discovery entities associated with the component broker, a set of native registries/lookup services, service provider components, data repositories, services and adapter entities. The component broker is analogous to an object request broker in other architectures. It is not a single entity, but is a collection of sub-entities. The component broker consists of the following sub-entities: the query handling entity, the domain security manager, the link manager and the adapter manager. A formal description of these entities involved in the URDS is given in chapter 2. The users of the URDS system can be the Component Assemblers, System developers or System Integrators and are responsible for developing a DCS based on client requirements. This research proposes a mobile agent-based version of the URDS, called MURDS (The UniFrame Mobile Agent-based Resource Discovery Service).

1.1.1 Motivation

The motivation for the MURDS is as follows:

The discovery of heterogeneous software components and their selection for a particular client request consists of two phases of the URDS. These phases are described below.

 Phase 1: Discovering heterogeneous software components

The Service Discovery Entity (SDE) and the Native Registry/Lookup Service (NR/LS) are the two participants that perform the dynamic discovery of heterogeneous software components in the URDS architecture.

The purpose of the SDE is to detect the presence of heterogeneous software components in the network, to register the functionality of these components in its data repository and to return a list of components that matches the requirements specified in the query for building a particular DCS.

The purpose of the NR/LS is to register components developed under a particular distributed component model such as Java RMI, CORBA, and .NET. The discovery

process is based on the propagation of multicast messages between the SDE and various Native Registries/Lookup Services (NRs/LSs). Hence, the functionality of these NRs/LSs is extended to listen and respond to multicast messages from the SDE.

Upon initialization, service provider components register their services with their NRs/LSs. Each SDE periodically multicasts its presence in the network. All the NRs/LSs, which listen for multicast messages, respond to the respective SDE multicast messages by passing their contact information. Service Discovery Entities (SDEs) query those NRs/LSs that respond to their announcements for the list of components registered with them. The NRs/LSs respond to the SDE by passing the list of components registered with them and the detailed service description of these components. The SDEs store this information in its data repository [9].

Phase 2: <u>Selecting appropriate components</u>

The SDE and the Query Handling Entity (QHE) are the two participants that are involved in the selection of appropriate software components based on the requirements provided by various clients.

The purpose of the QHE is to handle client requests by passing on their requirements to the SDEs. The SDEs process the requests that they receive from various Query Handling Entities (QHEs) and return the lists of software components that match the search criteria to the respective QHEs.

A client who wants to build a particular DCS submits a functional and QoS requirement specification to the system integrators/component assemblers. The system integrators/component assemblers contact the QHE and submit the client's request to retrieve a list of software components that match that request. The QHE, which has a list of SDEs, randomly picks a SDE as a 'Chief Service Discovery Entity' (CSDE) and delegates the job of selecting appropriate components from the remaining list of SDEs to the CSDE. The CSDE searches its data repository to find out components that matches the search criteria. Also, it selects a random subset of the remaining SDEs, delegating each SDE a list containing a portion of the remaining SDEs along with the query to be transmitted. The portion allocated is a ratio of the remaining SDEs to the number of SDEs in the chosen subset. Each of the subset SDEs is a CSDE and is responsible for

transmission of the query among the list of SDEs allocated to it and retrieval of the results back to the CSDE that spawned them. The transmission of the query includes selecting a subset of CSDEs and passing of the remaining SDE list and the query to the subset. The CSDE finally combines the result before sending it back to the QHE.

The periodic discovery of newly registered components involves repeated interactions between SDEs and NRs/LSs. The task of selecting appropriate service provider components based on particular search criteria involves multiple interactions between QHE and the list of SDEs. All the interactions are based on a request-reply protocol that is synchronous in nature. Therefore, these entities not only make several separate requests but also maintain network connections over an extended period of time until the corresponding requests are fulfilled. In the event of network failure, these entities would have to send their requests more than once to get a reply. This results in increased utilization of network and server side resources. Making use of asynchronous modes of communication and remote event notification relieves the respective entities from maintaining network connections over an extended period of time.

Also, an increase in the number of entities participating in the discovery phase increases the network resource utilization. If the available network bandwidth decreases, fewer entities would be able to discover components at a given point of time. This implies that the network bandwidth would become a bottleneck to the discovery process. A possible solution to avoid this scenario is to make the discovery mechanism less dependent on the network bandwidth. This can be achieved by reducing the number of transactions between the SDEs and the NRs/LSs as well as by making asynchronous calls.

Another issue that is related to the component discovery phase is as follows:

In the URDS, the nature of services provided by NRs/LSs is considered to be independent of the nature of the SDEs requesting services from them. A SDE requesting software components will be provided all the components that are available with NRs/LSs. An important fact that needs to be considered is to limit the services provided to the SDEs based upon a decision-making process that considers, among other things, the type of SDEs requesting the services. Various sets of distinguishing parameters, such as cost and security, can be used to determine the nature of the SDE groups. As these can

be heterogeneous by nature, an elegant mechanism is needed to seamlessly overcome these differences.

In order to address all of the above mentioned issues, this project evaluates the mobile agent paradigm as a possible enhancement of the UniFrame resource discovery mechanism. A mobile agent is a program that represents a user in a computer network and can migrate autonomously from node to node to perform some computation on behalf of the user [15].

According to the literature [16], mobile agents offer the following advantages

- They reduce the network load by sending the code to the data host instead of sending the data over the network.
- They overcome network latency by executing the code locally and hence they do not need a lot of bandwidth.
- They encapsulate protocols by carrying their own protocol code with them to the visiting host system.
- They execute asynchronously and autonomously, and are therefore independent of a continuously open network connection.
- They adapt dynamically to new environments.
- They are naturally heterogeneous and work well in a heterogeneous network.
- They are robust and fault-tolerant.

The introduction of mobile agents to discover software components and to select appropriate service provider components in the URDS architecture changes the interaction pattern between associated entities by sending mobile agents to execute their tasks locally, autonomously and asynchronously. Asynchronous mode of communication relieves the entities from maintaining network connections over an extended period of time. Local executions of the tasks reduce the consumption of network bandwidth to discover and to select components. In order to obtain component information from NRs/LSs, the SDEs must specify information about their identity or nature. A Mobile agent achieves this task by carrying information with it during the discovery process.

1.2   Objectives

The specific objectives of this project are:

- To provide a mobile agent-based architecture, MURDS, thereby enhancing the URDS architecture [9].
- To develop a prototype for MURDS using Java and Java-based agent development platform – Grasshopper.
- To validate the principles behind MURDS by conducting experiments on the MURDS prototype.

1.3   Contributions

The contributions of this project are:

- It provides a survey of the issues associated with the component discovery and component selection phases of the URDS and establishes the benefit of introducing mobile agents to address these issues in the MURDS architecture.
- It presents a framework for MURDS by adding mobile agents into the URDS architecture.
- It provides an access control model that allows mobile agents to get access to the resources associated with the NRs/LSs involved in the component discovery phase of the MURDS.
- It compares the perfomance of mobile agent based resource discovery service with the non-mobile agent based resource discovery service to prove that the mobile agent based resource discovery service provides better service when compared to the non-mobile agent based resource discovery service.

1.4   Organization of this report

This project report is organized into 6 chapters. The introduction of the project, along with problem definition and motivation, objectives and contributions were presented in this chapter. Chapter 2 provides related work. Chapter 3 provides an overview and design details. Chapter 4 provides implementation of the project. Chapter 5 describes the prototype validation by experimentation. Chapter 6 concludes this project with a discussion of what was accomplished and future work.

2. BACKGROUND AND RELATED WORK

This chapter starts with a brief overview of UniFrame, which is a basis for URDS and subsequently to MURDS. Section 2.2 describes various directory services and discovery services, whose shortcomings lead to the introduction of URDS for service discovery aspect in UniFrame. Section 2.3 gives an argument for mobility in the URDS architecture and a brief overview of mobile agent architecture. In order to implement mobile agents in MURDS, an agent implementation tool is required. Section 2.4 gives an overview of Grasshopper agent platform used for implementing mobile agents in MURDS.

2.1   Overview of UniFrame

UniFrame [6] is a framework for developing distributed computing systems based on an integration of heterogeneous software components that adhere to different distributed component models. The UniFrame consists of the Unified Meta-Component Model (UMM) and the UniFrame Approach (UA). The UMM described in [6] is the core part of UniFrame. The UA is a component based software engineering process based on UMM for creating a DCS out of available heterogeneous distributed software components. The following sub-sections describe UMM and UA.

2.1.1 The Unified Meta-Component Model (UMM)

UMM is divided into three parts namely Components, Service and Service Guarantees, and Infrastructure.

2.1.1.1 Components

UniFrame is a component-based framework where components form the building blocks of a DCS built out of it. Components, as defined in [8], are autonomous entities that adhere to different distributed component models and maintain a state, an identity

and a behavior with them. In addition, each component in UMM has three aspects namely Computational Aspect, Cooperative Aspect and Auxiliary Aspect.

The Computational Aspect, as described in [8], reflects the task(s) carried out by each component; the Cooperative Aspect of a component, as described in [8], indicates its interaction with other components; and the Auxiliary Aspect of a component, as described in [8], addresses the features required to build a distributed computing system such as mobility, security and fault tolerance.

2.1.1.2 Service and Service Guarantees

A Service, as defined in [8], could be a computational effort or an access to underlying resources. Any component that offers services must provide certain quality of service guarantees in order to be selected for the development of a particular DCS in the UniFrame. The quality of service of a component is an indication of its ability to carry out a specified service in spite of the constantly changing execution environment and a possibility of partial failures. The QoS offered by each component is dependent upon the computation performed, algorithm used, expected computational effort and resources required, the cost of each service, and the dynamics of supply and demand.

2.1.1.3 Infrastructure

UniFrame Resource Discovery Service (URDS) provides the necessary infrastructure to discover components as well as to select and integrate components that adhere to different component models in order to develop a distributed computing system. URDS infrastructure comprises of the following entities to carry out their specified tasks: a) Internet Component Broker (ICB) b) Headhunters (HHs), c) Meta-Repositories, d) Active Registries, e) Services (S1..Sn), and f) Adapter components (AC1...ACn). The following subsections give a description of all the entities specified in the URDS infrastructure. Figure 2.1 shows interaction of these components in the URDS architecture.

- *Internet Component Broker (ICB)*: The ICB, which was referred as a '*Component Broker*' in chapter 1, is not a single entity but a collection of the following services – *Query Manager (QM), the Domain Security Manager (DSM), Link Manager (LM),* and *Adapter Manager (AM).* The ICB acts as an all-pervasive component broker in an interconnected environment. It constitutes the communication infrastructure necessary to identify and locate services, enforce domain security and handle mediation between heterogeneous components. All of the services provided by an ICB are accessible at well-known addresses. It is expected that there will be a fixed number of ICBs deployed at well-known locations hosted by corporations or organizations supporting UniFrame.

  o *Domain Security Manager (DSM)*: The URDS discovery protocol is based on periodic multicast announcements. The multicast communication exposes URDS to various security threats such as eavesdropping, uncontrolled group access and masquerading. The DSM is responsible to provide security and integrity of multicast announcements that take place between respective entities in the URDS discovery process. The DSM handles the generation and distribution of secret keys for the ICB and enforces multicast group address and access control to multicast resources through authentication and use of access control lists.

  o *Query Manager (QM)*: The QM, which was referred as a '*Query Handling entity*' in chapter 1, is responsible to translate a system integrator's/component assembler's component requirement data into a structured query language (SQL) statement and to dispatch this query to the 'appropriate' HHs in order to receive a list of service provider components that match the search criteria specified in the query. 'Appropriate' HHs are selected based on the domain specified in the query. The QM and the LM are responsible for propagating the queries to other linked ICBs.

  o *Link Manager (LM)*: The LM is responsible to establish links between ICBs to form a federation and to propagate queries received from the QM to the linked ICBs. An ICB administrator configures the LM with the location information of other ICBs with which links are to be established.

- o *Adapter Manager (AM)*: The AM acts as a registry/lookup service for clients seeking adapter components. The adapter components register with the AM by specifying the component models that they can bridge efficiently. Clients contact AM to search for adapter components that match their requirements.

- *Headhunters (HHs)*: The HHS, which were referred to as '*Service Discovery entities*' in chapter 1, are responsible for detecting the presence of service providers, registering the functionality of these service providers and returning a list of service providers to the ICB that matches the requirements of the component assembler's/system integrator's request forwarded by the QM.

- *Meta-Repository (MR)*: The MR, which was referred as a '*data repository*' in chapter 1, is a database that is associated with a Headhunter to store the UniFrame specification information of exporters adhering to heterogeneous component models.

- *Active Registry (AR)*: The AR, which was referred as a '*native registry/lookup service*' in chapter 1, serves as a native registry/lookup service of a particular distributed computing model such as RMI, CORBA, .NET, etc., and is extended to listen and respond to multicast announcements from Headhunters. ARs also have introspection capabilities to discover not only the instances, but also the specifications of the components registered with them.

- *Services (S1…Sn)*: The services that are deployed on the network may be implemented in different distributed component models such as RMI, CORBA, .NET, etc. Each of these services identify themselves by the service type name and the XML description of the component's informal UMM specification.

- *Adapter Components (AC1…ACn)*: The ACs, which were referred to as '*adapter entities*' in chapter 1, are responsible to serve as bridges between components developed in different distributed component models.

- *Users (C1…Cn)*: The users of the URDS system can be Component Assemblers, System Integrators or System developers searching for services matching certain functional and non-functional requirements. However, in complete UniFrame, there will be no direct interaction between human users and the URDS. The interaction would be via the interface of the system integrator.
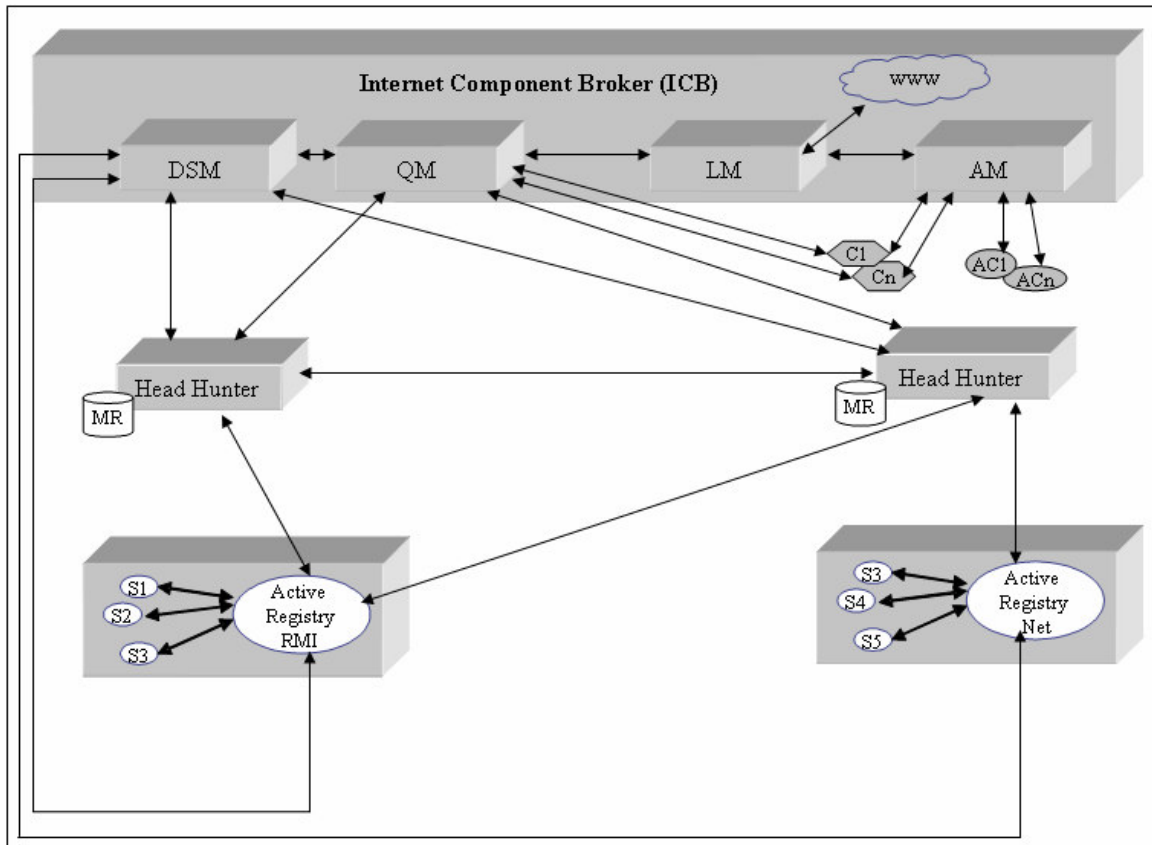
Figure 2.1 URDS Architecture (from [9])

The URDS architecture is organized as a federated hierarchy of ICBs and Headhunters in order to achieve scalability. Each ICB that is participating in the URDS consists of one level of Zero or more Headhunters attached to it and all ICBs are linked to one another with unidirectional links to form a federated group. The URDS discovery protocol is based on periodic multicast announcements. The URDS discovery process uses an administratively defined logical domain such as Financial Services, Health Care Services, etc. to locate services and these domains are determined by the organizations providing the URDS. The URDS architecture handles failures through periodic announcements (incase of headhunters), heartbeat probes (in case of link managers), and information caching.

2.1.2   The UniFrame Approach (UA)

The UniFrame Approach (UA) proposed in [6] is a technique for the automatic or semi-automatic generation of a DCS based on the integration of heterogeneous software components. The UA specifies two levels for the generation of a DCS namely Component Level and System Level. The Component Level allows component developers to create components based on UMM specifications, test and verify QoS of components and then deploy them on the network. The System Level allows system integrators/application programmers to select and generate a software solution for a particular DCS under consideration in an automatic or semi-automatic fashion to the maximum possible extent. The UMM specification, the UniFrame Quality of Service (UQOS) framework and the UniFrame System-Level Generative Programming Framework (USGDP) realizes the component level and the system level aspects of UA. The following sub sections give a brief description of the UMM specification, the UQOS and the USGPF.

2.1.2.1 UMM specification

Every component participating in the UniFrame must specify a set of QoS parameters based on UMM specification and it is the responsibility of a component developer to specify these parameters during the component development and deployment phase. Table 2.1 gives the UMM specification template for a component.

- Component Name: This entry specifies the name used to identify a component in the UMM specification.
- Component Subcase: This entry indicates information related to communication patterns of functions of the component.
- Domain Name: This entry specifies the domain scope for the component, for example, finance domain.
- System Name: This entry indicates the system family to which this component belongs to.

- Description: This entry provides an informal description of the services provided by the component. This information may include unique characteristics of the component that cannot be described in other entries.

UMM Specification

1.Component Name: <component name>
2.Component Subcase: <component subcase name>
3. Domain Name: <domain name>
4. System Name: <system family name>
5.Informal Description: <natural language description>
6. Computational Attributes:
    6.1 Inherent Attributes:
        6.1.1 id: <internet address for a concrete component, or N/A for an abstract component>
        6.1.2 Version: <version exression>
        6.1.3 Author: <developer name for a concrete component, or N/A for an abstract component>
        6.1.4 Date: <deployment time for a concrete component, or N/A for an abstract component>
        6.1.5 Validity: <valid time for a concrete component, or N/A for an abstract component>
        6.1.6 Atomicity: <Yes/No>
        6.1.7 Registration: <the registering headhunter for a concrete component, or N/A for an abstract component>
        6.1.8 Model: <component model for a concrete component, or N/A for an abstract component>
    6.2 Functional Attributes:
        6.2.1 Function description: <natural language description of component functions>
        6.2.2 Algorithm: <list of algorithms>
        6.2.3 Complexity: <component complexity for a concrete component, or N/A for an abstract component>
        6.2.4 Syntactic Contract:
            6.2.4.1 Provided Interface: <list of provided interfaces>
            6.2.4.2 Required Interface: <list of required interfaces>
        6.2.5 Technology: <technology name for a concrete component, or N/A for an abstract component>
        6.2.6 Expected Resources: <expected resources expression, NONE if not available for a concrete component, or N/A for an abstract component>
        6.2.7 Design Patterns: <list of used design patterns separated by a comma, or NONE>
        6.2.8 Known Usage: <list of known usage separated by a semi-colon, or NONE>

> 6.2.9 Alias: <list of aliases separated by a comma, or NONE>
>
>
> 7. Cooperation Attributes:
>   7.1 Preprocessing Collaborators: <list of Preprocessing Collaborators separated by comma, or NONE>
>   7.2 Postprocessing Collaborators: < list of Postprocessing Collaborators separated by comma, or NONE>
> 8. Auxiliary Attributes:
>   8.1 Mobility: <Yes/No>
>   8.2 Security: <security level>
>   8.3 Fault Tolerance: <fault tolerance level>
> 9. Quality of Service:
>   9.1 QoS Metrics: <list of QoS metrics separated by comma for an abstract component, or list of detailed QoS metrics separated by semicolon for a concrete component.
>   9.2 QoS Level: <level of QoS>
>   9.3 Cost: <compensation level>
>   9.4 Quality Level: <level of quality>

Table 2.1 UMM Specification Template (from [20])

- Computational Attributes: This entry describes the computational aspect of the component in terms of the following parameters.

  o Inherent Attributes:

    - ID: This is a unique string consisting of the host name and the port on which the component is running along with the name with which the component binds itself to a registry, for example: intrepid.cs.iupui.edu:8080/AccountServer.

    - Version: This entry indicates the version of the component.

    - Author: This entry indicates the authors of the component.

    - Date: This entry indicates the deployment time for a concrete component. It is not applicable for an abstract component.

    - Validity: This entry indicates whether a concrete component is valid. It is not applicable for an abstract component.

    - Atomicity: This entry indicates whether a component is atomic.

- Registration: This entry indicates the registration of a component with a particular headhunter participating in the UniFrame system. It is not applicable for an abstract component.

- Model: This entry indicates the component model that the component adhered to.

o Functional Attributes:

- Function Description: This entry provides a description of each of the functions supported by the component.

- Algorithm: This entry indicates the algorithms utilized by the component to implement its functionality if the type of the specification is concrete component. If the specification type is abstract component, then this entry means the corresponding concrete components must implement the indicated algorithms, e.g., Quick Sort.

- Complexity: This entry describes the order of complexity of the above-mentioned algorithms implemented by the component.

- Syntactic Contract: This entry provides the computational signature of the component's service interface. These interfaces are well defined in the process of generative domain engineering. Each component must specify its provided interfaces and required interfaces.

- Technology: This entry indicates the component technology utilized to implement the component, e.g., J2EE, CORBA etc.

- Expected Resources: This entry indicates the expected resources for the component, e.g., CPU, memory

- Design Patterns: This entry indicates the design patterns employed by the component.

- Known Usage: This entry indicates the known usages of the component.

- Alias: This entry indicates the alias names for the component.

- Cooperation Attributes
  - Preprocessing Collaborators: This entry indicates the dependency of this component on other components.
  - Postprocessing Collaborators: This entry indicates other components that may depend on this component.
- Auxiliary Attributes
  - Mobility: This entry indicates whether the component is mobile or not.
  - Security: This entry indicates the security level of the component.
  - Fault Tolerance: This entry indicates the fault tolerance level of the component.
- Quality of Service
  - QoS Metrics: Each abstract component should list the QoS metrics that should be provided by the implementation components (Concrete components). For a concrete component, provided information for each QoS metrics includes: a) QoS parameter name, b) type of parameter: static/dynamic, c) min/max limit. If the QoS metric is dynamic, also provide information about: d) environment values for the min/max ratings, and e) variation in parameter values according to environment.
  - QoS Level: A component developer may offer several possible levels of QoS. This entry is not applicable to an abstract component.
  - Cost: This entry indicates the compensation level for the component.
  - Quality Level: This entry provides an overall assessment of a concrete component. It is not applicable to an abstract component.

During the component development and deployment phase, the natural language specification is converted into a standard XML-based specification, which can be automatically discovered by URDS.

2.1.2.2 The UniFrame QoS Framework (UQOS)

The concepts of the service and service guarantees are an integral part of every component in UMM and they also pay an important role in the system generation phase

of the UniFrame. The UniFrame QOS (UQOS) framework is an implementation of the services and service guarantees aspect of the UMM.

In order to utilize the Service and Service guarantees of UMM to assure the QoS of a DCS, following issues have to be addressed: a) a framework to objectively quantify the QoS of software components, b) a standardized QoS catalog for reference by software component developers and application engineers, c) a standard to incorporate the effect of the environment on the QoS of software components into the component development process, d) a standard approach to incorporate the effect of usage patterns on the QoS of software components into the component development process, and e) a QoS specification scheme to specify the QOS of software components. The UQOS framework consists of four parts to solve these issues:

- The QoS catalog.
- The approach for accounting the effect of the environment on the QoS of software components.
- The approach for accounting the effect of usage patterns on the QoS of software components.
- The specification of the QoS of software components.

Detailed description of these parts can be found at [21, 22, 23].

2.1.2.3 The UniFrame System Level Generative Programming Framework (USGPF)

The QoS is an integral part of every component in UMM and is inherent in any system generated from these components. Thus, the QoS plays an important role in the entire UniFrame approach and helps to create QoS-aware DCS from heterogeneous distributed software components. The UniFrame approach also shifts from the traditional software development paradigm of developing single DCS to the paradigm of developing a DCS family.

The USGPF [20] realizes the UniFrame Approach on the system level. More specifically, it addresses the generative domain engineering and the generative application engineering aspects of the software development process in the UniFrame approach. The USGPF is composed of three parts:

- The UniFrame Generative Domain Model (UGDM), which defines the common and variable properties of a DCS family.

- The UniFrame UGDM Development process (UGDP), which defines the procedure to efficiently create a UGDM for a DCS family with QoS constraints.

- The UniFrame System Generation Framework (USGI), which facilitates the automatic generation of QoS-aware DCS from a DCS family by integrating heterogeneous software components.

  Detailed information on UGDM, UGDP and USGI can be found at [20].

The above discussion shows that the URDS provides a unique feature of automatic or semi-automatic generation of a DCS by composing components belonging to different distributed computing models. Most of the discovery services that are available in the market does not provide this feature and consider components belonging to a particular model. The following section provides a survey of various directory services and discovery services that are available in the market.

2.2   Resource Discovery protocols

Resource discovery refers to the process of identifying resources on a network and making these resources available to users and applications. Resources are defined to be any piece of hardware or software that provides a service to users and applications. Resource discovery protocols are classified into two categories namely Directory services and Discovery services. The following subsections provide a brief description of different resource discovery protocols available under these two categories.

2.2.1 Directory Services

A service that stores collection of bindings between names and attributes and that looks up entries that match attribute-based specification is called a directory service [24]. Directory services are also called yellow pages services. Some of the discovery protocols that come under Directory Services are Universal Description, Discovery and Integration

(UDDI) [10], CORBA Trader Services [11], Light Weight Directory Access Protocol (LDAP) [12], X.500 [13], Domain Name Service (DNS) [14], and Global Name Service (GNS) [24]. The following sub-sections provide a brief overview of these services.

2.2.1.1 Universal Description, Discovery and Integration (UDDI)

Universal Description, Discovery and Integration (UDDI) [10] specification defines a standard way to describe, to publish and to discover information about web services developed by different business providers. The term "web service" refers to "specific business functionality exposed by a company, usually through an internet connection, for the purpose of providing a way for another company or software program to use the service" [10]. The UDDI architecture uses Simple Object Access Protocol (SOAP), which is built on top of Extensible Markup Language (XML), to allow a program invoke service interfaces across the Internet in a language independent and distributed manner. SOAP defines a simple way to package information for exchange across system boundaries. The UDDI architecture consists of three components namely Web Service Providers, Service Requesters and Service Brokers and it follows a centralized model where service providers register their services to a common UDDI service registry or service broker, service requestors search for services at service registry using Web Service Description Language (WSDL).

2.2.1.2 CORBA Trader Services

The CORBA Trader service [11] serves as a "yellow pages" lookup service for service providers and service consumers in the distributed computing environment. It consists of three key components namely Exporters, Importers and Traders. The Exporters, which are service provider objects, advertise their services to traders. The Traders, which act as lookup services, register service description as well as service location of services advertised by exporters in their directory. The Importers, which are service consumers, specify their search criteria to their local Trader to retrieve a list of

services registered with it or make a request for a particular service by providing service description to it. The local Trader searches in its local directory and also propagates the query to other traders participating in the CORBA trading service to find a service matching the search criteria. Traders are defined as CORBA interfaces, and all advertisements, requests, and replies are CORBA objects. CORBA interfaces are defined using Interface Definition Language. Since the Traders depend on CORBA, all participants must be cast as CORBA objects and use CORBA protocols.

2.2.1.3 Light Weight Directory Access Protocol (LDAP)

The LDAP [12] is a lightweight version of the Directory Access Protocol and is a part of X.500, which is a standard for directory service in a network. It defines a lightweight access mechanism in which clients send requests to and receive responses from LDAP servers. The LDAP information model is a directory service that follows a scalable hierarchical tree like structure and each node in the tree contains information about some object in terms of an entry, which is analogous to a record in a relational database. Each entry carries some attributes where each attribute consists of types with one or more values. The attribute type describes what the information is about and the attribute value specifies the actual information in text format. LDAP's query model allows for search and retrieval of entries stored in the LDAP directory server. Since LDAP is organized as a hierarchy, the queries can be limited to particular parts of the hierarchy. Though the design is scalable queries over very large domains are likely to be very inefficient. LDAP does not have any built in security model and relies on other network services for this purpose. LDAP does not specify protocols for "spontaneous" discovery and because of the complexity; it may not be well suited for either near real-time discovery, or for very large numbers of services.

2.2.1.4 Global Name Service (GNS) and Domain Name Service (DNS)

Global Name Service (GNS) [24] was designed and implemented by Lampson and colleagues at the DEC Systems Research Center to provide facilities for resource

location, mail addressing and authentication. GNS manages a naming database that is composed of a tree of directories holding name and values. Directories are named by multi-part pathnames referred to a root, or relative to a working directory, much like filenames in a UNIX file system.

Domain Name Service (DNS) [14] is an Internet service that translates domain names into IP addresses. The DNS protocol provides static database of name-address maps, which is hierarchically partitioned. The naming data is replicated and cached in order to achieve scalability. Recent extensions to DNS support a very limited set of service types and a few attributes that can be used to search. The types of queries supported by DNS include host name resolution and reverse resolution, mail host location, host information and well-known services information [19]. DNS is a trusted service, security is provided by controlling access to a few privileged users. Arbitrary user applications may not add or modify the DNS database.

2.2.2 Discovery Services

A discovery service is a directory service that registers the services provided in a spontaneous networking environment [24]. In spontaneous networks, devices are liable to connect without warning and without administrative preparation. Some of the discovery protocols that come under Discovery Services are JINI [25], Service Location Protocol (SLP) [26,27], Ninja Project: Secure Service Discovery Service (SSDS) [28,29], Salutation [30,31], Universal Plug and Play (UPnP) [32,33], Bluetooth Service Discovery Protocol [34], DReggie [31] and UniFrame Resource Discovery Service (URDS) [9]. The following sub-sections provide a brief overview of these services.

2.2.2.1 JINI

JINI is a distributed service architecture developed by Sun Microsystems [25]. JINI is developed in Java and uses Java Remote Method Invocation (RMI) to achieve communication among services and clients. Service, Client and Lookup Service are the key components involved in the discovery related activities of JINI. A service represents

a hardware device, a software program or their combination utilizing the Java language. A service registers the "service object" or "service proxy" associated with it at the lookup service. A client who is looking for a particular service contacts the lookup service to get the list of services available with it. A lookup service acts as a directory to both services and clients to register and locate services respectively. JINI's discovery protocol provides multicasting and directed modes of operation to its key components to locate other relevant components in the network.

2.2.2.2 Service Location Protocol (SLP)

Service Location Protocol (SLP) is an Internet Engineering Task Force (IETF) standard for dynamically discovering network resources [26,27]. SLP discovery mechanism uses a set of predefined attributes to describe both software and hardware services. SLP architecture consists of three types of agents namely User Agents (UA), Service Agents (SA) and Directory Agents (DA). UAs are responsible for discovering resources on behalf of clients that request services, SAs are responsible to advertise available services to DAs, and DAs are responsible to maintain the list of all services advertised by service agents and to respond to user agent requests.

SLP provides Active Discovery, Passive Discovery and Dynamic Host Configuration Protocol (DHCP) mechanisms to UAs and SAs to discover directory agents in the network. In Active Discovery, UAs and SAs use multicasting to send SLP requests to discover DA in the network where as in Passive Discovery, DAs periodically multicast its presence to UAs and SAs in the network. In Dynamic Host Configuration Protocol (DHCP)[14], UAs and SAs can use DHCP servers that are configured to distribute Directory Agents information to the sources that request the information.

SLP uses the following two modes of operation depending upon the availability of a DA on the network:

When a DA is available on the network, SAs and DAs multicast their messages to a well-known multicast address to locate DAs and upon receiving unicast messages in response from DAs, SAs register their service with the DA by sending a registration message where as UAs send a service request message when a client needs a service.

When a DA is unavailable on the network, UAs multicast their service messages to a well-known multicast address and SAs send a unicast message when a match is found.

2.2.2.3 Ninja Project: Secure Service Discovery Service (SSDS)

The SSDS is a component of the Ninja research project at University of California, Berkeley [28, 29]. It is similar to other discovery protocols but provides significant improvements in reliability, scalability and security. SSDS is implemented in Java and uses XML for service description and location, rather than java objects.

The main components of SSDS architecture are Service Discovery Service (SDS) servers, services, capability managers, certificate authorities and clients. SDS servers periodically multicast authenticated messages containing a list of domains that they are responsible for sending service announcements and caches the service descriptions that are advertised in the domain.

Services participating in the SDS system continuously listen for SDS server announcements to determine the appropriate SDS server for its service descriptions and multicasts its service descriptions to the multicast address using authenticated, encrypted one-way service broadcasts. Clients listen to a well-known SDS global multicast address to identify SDS servers related to their domains and submit a query in the form of an XML based service description.

The SDS uses certificates signed by a well-known certificate authority to authenticate between principles and their public keys. It uses capabilities generated and distributed by capability managers as an access control mechanism to enable services to control the set of users that are allowed to discover their existence.

2.2.2.4 Salutation

Salutation [30, 31] is an open standard, communication, operating system and platform-independent service discovery and session management protocol. The architecture provides a standard method for applications, services and devices to describe

and to advertise their capabilities to other applications, services and devices. The architecture defines three components namely client, server and Salutation Lookup Manager (SLM). The SLM serves as a service broker for services in the network and it classifies services into a collection of Functional Units (FU) where each functional unit represents some essential feature such as fax, print, scan, etc. The SLM uses Sun's RPC to communicate with other SLMs and it can be discovered by both unicast and broadcast methods. When a client submits a query for a particular service to the local SLM, it searches the SLM directory associated with it for a list of services based on a comparison of the required service type specified in the query and also propagates the query to other SLMs, where one SLM is a client to another SLM.

2.2.2.5 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) [32, 33] is a standard for spontaneous discovery from Microsoft Corporation. UPnP uses Simple Service Discovery Protocol (SSDP) [34] to discover services in the network. SSDP operates on top of the existing open standard protocols utilizing HTTP over both unicast (HTTPU) and multicast UDP (HHTPMU). UPnP uses XML to describe service features and capabilities. When a service wants to join the network, it sends out a multicast message to advertise its services to lookup services. If a lookup service is available in the network, it can record this advertisement to be subsequently used to satisfy the client's service discovery requests. Additionally, each service on the network may also observe these advertisements. When a client wants to discover a service, it can either contact the service directly through the URL stored in the service advertisement or it can send out a multicast query message to lookup service and it can receive a unicast reply from either a lookup service or the service itself.

2.2.2.6 Bluetooth Service Discovery Protocol

Bluetooth [35] is a short-range wireless technology that allows devices to exchange data and voice in real time. It consists of a set of protocols that constitute the

protocol stack. Service Discovery Protocol is one of the protocols in the protocol stack, and it provides efficient service discovery on resource-constrained devices.

2.2.2.7 DReggie

DReggie [36] is a Jini-based Semantic Service Discovery System under development at the University Of Maryland, Baltimore County. It attempts to take Jini and similar service discovery systems beyond their simple syntax-based service matching techniques by adding semantic matching capabilities to the service description facilities. Semantic service matching widens the scope of a certain service discovery request by being able to locate services based on the functional description of the service. DReggie uses DARPA Agent Markup Language (DAML) [37] and intelligent reasoning modules to carryout semantic matching process. DReggie enhanced Jini Lookup Service (JLS) to enable discovery of Jini-enabled services. Using DReggie, clients can discover services in a manner unchanged from the existing Jini Lookup and Discovery infrastructure. Any service that registers with DReggie's enhanced Jini Lookup Service uses DAML to describe its capabilities, requirements and service attributes. Any client who wants to find a service uses DAML to describe service requirements along with its constraints. The DReggie lookup server contains two different modules to carryout the matching process of the service request with descriptions of the advertised services: a simple java-based matching module and an advanced prolog-based reasoning module. The initial implementation uses simple java based matching module to find either exact or approximate service matches.

2.2.2.8 Grid Discovery Service (Globus Toolkit MDS)

The Monitoring and Discovery Service (MDS) incorporated in Globus Toolkit consists of two components namely the Grid Resource Information Service (GRIS) and the Grid Index Information Service. The GRIS runs on resources deployed on the Grid and is an information provider framework for specific information sources. At a higher level, the GIIS is a user accessible directory server that accepts information from child

GIIS and GRIS instances and aggregates this information into a unified space. MDS also supports searching for resources by characteristics. The discovery service is mainly employed for computational resources deployed on the Grid. Unlike URDS, the performance of a query to the MDS cannot be predicted with a pre-defined formula [47] and is depended on the complexity of the associated hierarchy of GRIS and GIIS.

2.2.3 Characteristics of Resource Discovery protocols

All the problems that the discovery protocols discussed under sections 2.2.1 and 2.2.2 aim to solve can be summarized into Service Advertisement, Service Request and Match Making.

- *Service Advertisement*: Service Providers advertise their services by providing their availability, contact information and other necessary information. This advertisement could be in the form either registering with a directory service or through multicast or broadcast communication.

- *Service Request*: Service Consumers searching for services forward their request to some well-known directory service or send request across the network through the process of multicast or broadcast communication.

- *Match Making*: Match Making is the process in which service producers and service consumers hookup. This process can be facilitated through a directory service or direct discovery between producers and consumers.

Issues related to reliability and scalability of all these discovery protocols are handled either through a hierarchical or federated organization. All the discovery and directory protocols exhibit some common features. A comparison of these features is provided in table 2.1 based on the information provided in [50].

Feature List	Directory Services	Discovery Services
Information Storage and Retrieval	Storage of information in static databases. For greater scalability, information can be replicated on multiple	Spontaneous discovery and configuration of network services and devices. Some Discovery services such as

	static databases.	SSDS, JINI maintain caches of information about discovered services and devices and update caches periodically to reflect the global state of the system.
Failure Detection	Do not monitor the availability of resources or their failure due to external circumstances such and node/link failure, etc. They usually do not possess any event generation mechanisms either to inform clients of resource registration or withdrawal.	Automatically configure according to service availability.
Management	Centralized control. Usually maintained by privileged administrators.	Decentralized management with limited administration.
Search Semantics	Usually provide lower flexibility with respect to the search criteria that can be specified for service selection.	Allows for selection of very specific types of service.
Interoperability	Do not provide interoperability of services that belong to different models.	Some services enable interoperability of services that belong to different models through the use of bridges and proxies.

Table 2.2 Comparison of the Features of the Directory and Discovery services

The Directory and Discovery Services described under sections 2.2.1 and 2.2.2 are mostly designed for 'closed' systems, i.e., systems, although distributed in nature, are developed and deployed in a confined setup. Such systems do not take advantage of the heterogeneity, local autonomy and the open architecture that are characteristic of DCS. The URDS architecture and the MURDS architecture on the other hand are designed for 'open' systems by providing for the discovery and interoperation of distributed heterogeneous software components. The MURDS uses mobile agents to reduce the message consumption and to interoperate with ARs to discover components. The following section provides an overview of agents.

2.3 Overview of Agents

2.3.1 Agent

An agent is a computational entity that
- Acts on behalf of other entities in an autonomous fashion,
- Performs its action with some level of proactivity and/or reactiveness, and
- Exhibits some level of the key attributes of learning, co-operation and mobility [33].

2.3.2 Mobile Agent

A mobile agent is a software entity that represents a user in a computer network and can migrate autonomously from node to node to perform some computation on behalf of the user [15]. Mobile agents inherit the characteristics of an agent described in section 2.3.1.

2.4 Overview of models in an agent

A mobile agent exists in a software environment called mobile agent execution environment. It is distributed over a network of heterogeneous computers [38] whose primary task is to provide an environment in which mobile agents can execute. A mobile agent must contain an agent model, a life-cycle model, a computational model, a security

model and a navigation model to work in the distributed computing environment. The following subsections give a description of these models.

2.4.1 Agent Model

An Agent model defines "the internal structure of the intelligent agent part of a mobile agent" [38]. The structure of this model defines the autonomy, learning and co-operative characteristics of an agent and also specifies the reactive and proactive nature of agents.

2.4.2 Life-cycle Model

A Life-cycle model defines "the different execution states of a mobile agent and the events that cause the movement from one state to another" [38]. The most prominent life cycle models that different mobile agent tools exhibit are the persistent process model and the task-based model.

- Persistent process model

  The persistent process model consists of a 'start' state, a 'running' state, a 'frozen' state and a 'death' state. Upon creation, a mobile agent enters a 'start' state, executes a persistent process in a 'running' state and enters a 'death' state when the process is terminated. When an agent is transported from one node to another node, the process in the running state is check-pointed and the agent enters a 'frozen' state. Then, its context is delivered to the destination node where the process is resumed and re-enters the 'running' state at the point it left off.
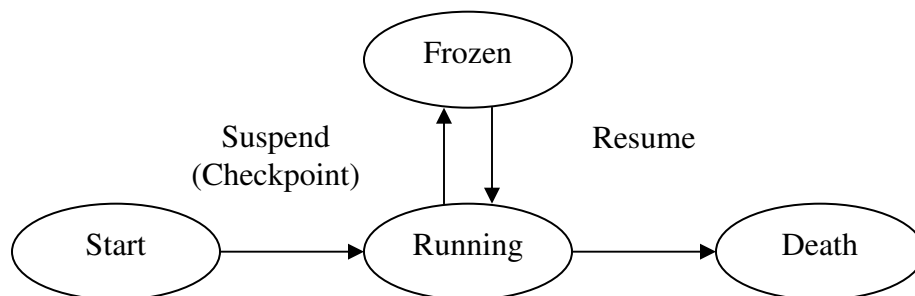
Figure 2.2 Persistent process based life cycle

The persistent process model is considered to be the most flexible life cycle as all other life cycle models can be built on top of this model [38]. Telescript [39], AgentTCL [41], Tryllian's Agent Development Kit [42] are some of the mobile agent tools that follow a persistent process model.

- Task based model

The task-based model consists of a 'start' state, a group of tasks and a 'death' state. Upon creation, a mobile agent enters a 'start' state. Then depending upon a set of conditions, it executes appropriate tasks where each task has its own state and finally enters the 'death' state upon the completion of execution of all appropriate tasks. In this model, the mobile agent loses its context of the currently executing task when it moves from one host to another host but stores the task that has to be started when it enters a
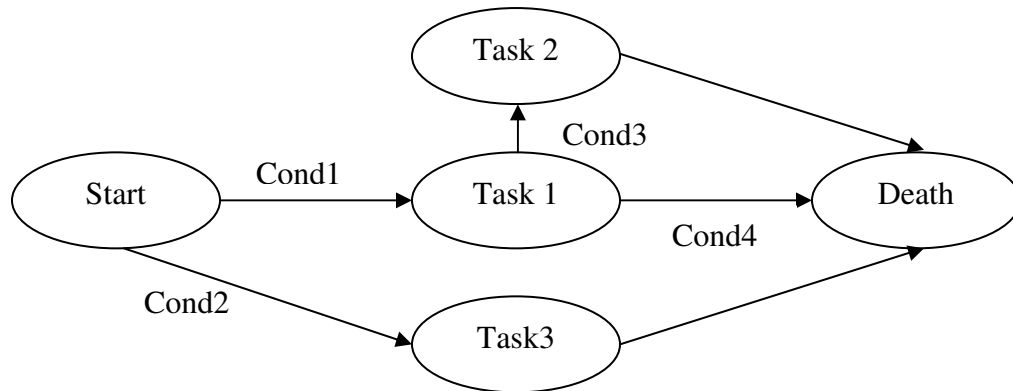
Figure 2.3 Task based life cycle

new host. The flexibility of this approach is reduced because of the loss of context information during transport [38]. Aglets [43], Voyager [44], Ajanta [45], Grasshopper [40], and JADE [46] are some of the mobile agent tools that follow task-based model.

2.4.3 Computational Model

A computational model defines "how a mobile agent executes when it is in a 'running' state". The computation takes place in an environment and is facilitated by some

form of processor. A processor could be the CPU of a computer or a more abstract processor as can be found in the Java virtual machine.

2.4.4 Security Model

Mobile agents are software entities that are independent of computer and transport layer and are dependent only on the agent execution environment. The autonomous behavior of mobile agents and the malicious environment of the Internet give rise to various important security issues for both the software agent and its execution environment. Hence, a security model deals with the protection of host nodes and mobile agents from one another. Security issues related to the mobile agent environment are broadly classified into two areas namely 'threats for agent hosts' and 'threats for agents'. The following subsections give a description of these threats.

- Threats to host systems from malicious agents

    Mobile agents run in an open distributed environment. An agent platform serves as an agent execution environment for mobile agents on a host system. It allows mobile agents to access host resources such as file system, system memory, local executable code, peripherals, and CPU cycles. This allows mobile agents to perform their tasks without human interventions. However, a mobile agent's ability will put the host platform at risk if an agent becomes malicious. Therefore, hosts in an open distributed environment may encounter a variety of security threats due to the execution of malicious agents. These security threats are classified into the following categories [17]:

    o Leakage: acquisition of data by an unauthorized party.

    o Tampering: alteration of data by an unauthorized party.

    o Resource stealing: use of facilities by an unauthorized party.

    o Vandalism: malicious interference with a host's data or facilities with no clear profit to the perpetrator.

    More specifically, a mobile agent may choose any of the traditional methods of attack like eavesdropping, masquerading, message tampering, message replay and viruses to harm a host system. Therefore, host systems use standard techniques such as

cryptography, authentication, digital signatures and trust hierarchies to guard host resources.

- Threats to mobile agents from malicious host systems

  As an independent and autonomous program, a mobile agent migrates between nodes in a heterogeneous network and performs tasks on behalf of its owner. During its course of travel, a mobile agent interacts with different components like host systems, other network entities as well as other mobile and stationary agents. Some of these components may be malicious and may pose security threats to the mobile agent. Security threats to a mobile agent are classified into the following categories [18]:

  o Integrity attacks: Violation of the integrity of mobile agent due to tampering of agent's code, state or data.

  o Availability refusal: Delaying the allocation of resources or preventing the access of specified objects to an authorized mobile agent to carry out its task on a specified host.

  o Confidentiality attacks: Violation of the privacy of a mobile agent due to the illegal access or disposal of mobile agent resources by the host environment.

  o Authentication risks: Jeopardizing the intended goal of a mobile agent by providing false identity.

  Trust based computing, recoding and tracking, cryptography and time based techniques are the countermeasures suggested to avoid some of the above specified attacks on mobile agents. According to the literature [18], most of the available countermeasures focus on 'integrity attacks' and very few measures exist to counter 'availability refusals' or 'authentication risks'.

2.4.5 Communication Model

A communication model deals with the interaction of mobile agents with entities like users, static or mobile agents, host execution environment, etc.

Mobile agents need protocols to communicate with all of these entities. A protocol is an implementation of a communicating model [38]. Distributed computing

environments use a wide variety of protocols for different purposes. Because of this reason, mobile agents need more than one communication model to communicate with different entities.

2.4.6 Navigation Model

A navigation model deals with all aspects of agent mobility from the discovery and resolution of destination hosts to the manner in which a mobile agent is transported [39].

Since mobile agents can execute only in a software environment, the mobile agent execution environment implements most of the above-mentioned models. It may also provide support services that relate to the mobile agent environment itself, support services pertaining to the environments on which the mobile agent environment is built, services to support access to other mobile agent systems, and finally support for openness when accessing non-agent-based software environments [38].

The introduction of mobile agents to discover heterogeneous software components in MURDS requires a software tool that serves as a mobile agent platform during the prototype implementation phase. Though the MURDS architecture that is described in chapter 3 did not adhere to any specific implementation methodology, the MURDS prototype implementation uses java and java-based technologies. Therefore, the following section provides an overview of a java-based mobile agent platform, Grasshopper, which serves as the mobile agent platform in the prototype implementation of MURDS.

2.5 Overview of Grasshopper

Grasshopper [40] is a mobile agent platform that is built on top of a distributed processing environment. By using Grasshopper, an integration of the traditional client/server paradigm and mobile agent technology can be achieved. Grasshopper is used to implement mobile agents in MURDS. Distributed Agent Environment (DAE), which provides the mobile agent environment in Grasshopper, is composed of the following entities - regions, places, agencies and different types of agents. Figure 2.3 shows an

abstract view of these entities in the DAE of Grasshopper, and the following subsections give a description of each of these entities.
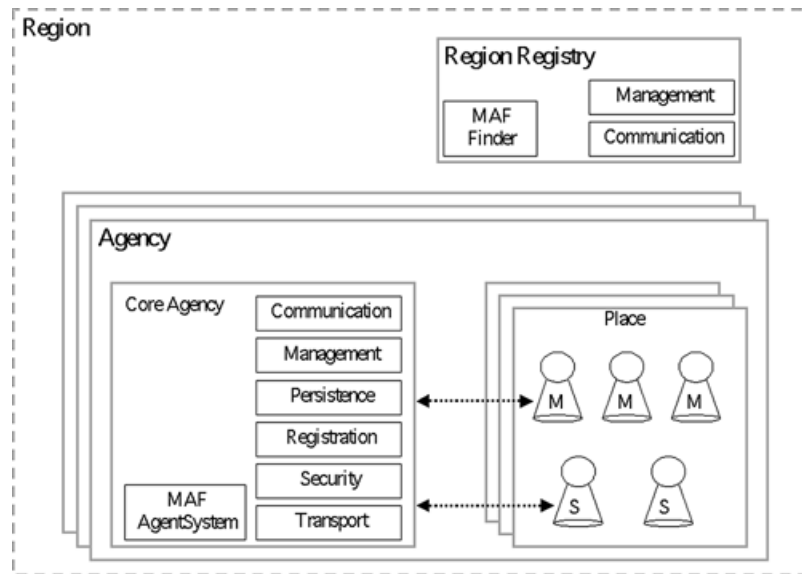
Figure 2.4 Abstract view of entities in the Distributed agent environment of Grasshopper (from [40], with the permission from IKV++ Technologies)

2.5.1 <u>Agents</u>

Taking the most commonly accepted attribute of an agent into consideration, Grasshopper defines 'an agent as a computer program that acts autonomously on behalf of a person or organization'. Grasshopper presents two types of agents in its context, namely, mobile agents and stationary agents.

- Mobile Agents

Mobile agents in Grasshopper are able to move from one physical network location to another. In this way, they can be regarded as an alternative or enhancement of the traditional client/server paradigm. While client/server technology relies on remote procedure calls across a network, mobile agents can migrate to the desired communication peer and take advantage of local interactions. In this way, several

advantages can be achieved, such as reduction of network traffic or a reduction of the dependency of network availability.

- Stationary agents

    Stationary agents do not have the ability to migrate actively between different network locations. Instead, they are associated with one specific location.

2.5.2 Agency

An agency is the actual runtime environment for mobile and stationary agents. At least one agency must run on each host that shall be able to support the execution of agents. A Grasshopper agency consists of two parts namely the Core Agency and one or more Places. The following subsections give a description of the Core Agency and the Place.

2.5.2.1 The Core Agency

The Core Agency represents the minimal functionality required by an agency in order to support the execution of agents. It provides the following services for the execution of agents in the DAE.

- Communication Service

    This service is responsible for all remote interactions that take place between the distributed components of Grasshopper, such as location-transparent inter-agent communication, agent transport, and the localization of agents by means of the region registry.

- Registration Service

    Each agency must be able to know about all agents and places currently hosted, on the one hand for external management purposes and, on the other hand, in order to deliver information about registered entities to hosted agents

- Management Service

    The management services allow the monitoring and control of agents and places of an agency by (human) users. It is possible, among others, to create, remove, suspend and resume agents, services and places, to get information about specific agents and services, to list all agents residing in a specific place and to list all places of an agency.

- Transport service

    The transport service supports the migration of agents from one agency to another. It handles the externalization and internalization of agents, and the coordination of the actual transfer that is performed by the communication service.

- Security Service

    Grasshopper supports two security mechanisms namely 'external' and 'internal' security. External security protects remote interactions between the distributed Grasshopper components, i.e. between agencies and region registries. On the other hand, internal security protects agency resources from unauthorized access by agents.

- Persistence Service

    The Grasshopper persistence service enables the storage of agents and places on a persistent medium. This way, it is possible to recover agents or places when needed, e.g. when an agency is restarted after a system crash.

    Detailed description of these models can be found at [35].

2.5.2.2 Place

    A place provides a logical grouping of functionality inside an agency. For example, there may be a communication place offering sophisticated communication features, or there may be a trading place where agents offer or buy information or service access. Therefore, the name of a place should reflect its purpose.

2.6 Region

The region concept facilitates the management of the distributed components in the Grasshopper environment, i.e., - agencies, places, and agents. Agencies as well as their places can be associated with a specific region by registering themselves within the accompanying region registry. Each registry automatically registers each agent that is currently hosted by an agency associated with the region. If an agency moves to another location, the corresponding registry information is automatically updated.

2.7 Region registry

A region registry maintains information about all components that are associated with a specific region. When a new component (i.e., an agency, place or agent) is created, it is automatically registered within the corresponding region registry. While agencies and their places are associated with a single region for their entire lifetime, mobile agents are able to move between the agencies of different regions. After each migration, the current location of mobile agents in which they are residing is updated in the corresponding region registry. By contacting the region registry, other entities (e.g. Agents, human users) are able to locate agents, places and agencies residing in a region. Besides, a region registry facilitates the connection establishment between agencies or agents.

This Chapter provided an overview of the Unified Meta-Model Framework (UniFrame), discovery based services, agents and Grasshopper. The next chapter provides the MURDS architecture, its design and implementation.

3. MURDS ARCHITECTURE

This chapter provides architecture of the MURDS. The MURDS architecture is an enhanced version of the URDS architecture. The MURDS focuses on the issues related to the component discovery and component selection mechanisms used in the URDS. The MURDS architecture modifies URDS architecture by replacing the request-reply communication with the mobile agent-based communication. The following paragraphs provide a review of the component discovery and the component selection mechanisms used in the URDS [9].

The Headhunters (HHs) and the Active Registries (ARs) are the two entities that are involved in the dynamic discovery of heterogeneous software components in the URDS. The discovery of components is carried out within an administratively scoped domain where each domain refers to an industry sector, such as Financial Services or Health Care Services, and is supported by the sector or organization providing the URDS service. Therefore, all the ARs and the HHs that are participating in the discovery process belong to a particular domain. The discovery of components is carried out with the help of multicast communication between the HHs and the ARs. In order to find out components that belong to a particular domain, the HHs periodically multicast their presence to a multicast group. The ARs, which belong to the same multicast group, listen for these messages and send unicast messages that contain their contact information to all the respective HHs. The HHs query those ARs that respond to their multicast messages for the UniFrame specification of all the components registered with them. The ARs respond to the HHs by sending the list of components registered with them along with their UniFrame specification. The HHs store the information of all these components in their local Meta-Repository (MR) and uses this information during the component selection process to select appropriate components that match client requirements.

The HHs and the Query Manager (QM) are the two entities involved in the component selection process of the URDS. When a client submits a functional and QoS requirement specification to the system integrators/component assemblers, they contact the QM and specify the search criteria to it.   The QM contacts the

DomainSecurityManager (DSM) and gets the list of HHs that belong to the domain specified in the search criteria. Then the QM randomly picks a HH termed as the 'Primary Headhunter' (PH) and delegates the job of selecting components from the remaining list of HHs to the PH. The PH first searches its local MR to find out components that match the search criteria. Then, it selects a random subset of the remaining HHs, delegating each HH a list containing a portion of the remaining HHs along with the query to be transmitted. The portion allocated is a ratio of the remaining HHs to the number of HHs in the chosen subset. Each of the subset HHs is a PH and is responsible for transmission of the query among the list of HHs allocated to it and retrieval of the results back to the HH that spawned them. The transmission of the query includes selecting a subset of HHs and passing the remaining HH list and the query to the subset. The PH finally combines the results it receives before sending them back to the QM.

The following sub-section provides a detailed description of component discovery phase and component selection phase in the MURDS.

3.1 Component Discovery and Component Selection phases

Phase 1: Discovering heterogeneous software components

The Active Registry, the Headhunter and a mobile agent acting on behalf of the Headhunter are the main entities that are involved in the dynamic discovery of heterogeneous software components.

Heterogeneous software components, which are available on the network, may offer services that belong to different application domains. Also, the entities that are involved in the discovery process in the MURDS (i.e., The Headhunter and the Active Registry) are associated with a domain. Hence, the mobile agent that acts on behalf of a Headhunter visits only those Active Registries that belong to the same domain as the Headhunter. Each principal that participates in the discovery process would be assigned to a particular domain by the system administrator. Upon initialization, each principal (i.e., the AR and the HH) contacts the DSM with its authentication credentials to participate in the discovery process. The DSM validates authenticity of the credentials

and registers the contacted principal with itself. Periodically, all the contacted principals update their availability with the DSM. Once deployed on the network, heterogeneous software components register services with their respective ARs. Each HH periodically contacts the DSM to get the list of ARs registered with the DSM. Upon receiving this list, each HH creates a mobile agent and sends it across the network to discover components that are newly registered with each AR specified in the list. The mobile agent randomly picks one AR at a time from the list, moves to the location of that AR and contacts it locally to get the list of components along with their detailed UniFrame specifications. The mobile agent repeats this process until it visits all the ARs specified in the list and returns the list of components to its respective HH. Upon receiving the information from the mobile agent, the HH stores the component information in its local MR and removes the mobile agent from the system. The HH uses the information available in its local MR to select appropriate components during the component selection process.

Phase 2: <u>Selecting appropriate components</u>

The HH, the QM and a mobile agent acting on behalf of the QM are the main entities that are involved in the selection of components that match client requirements.

A client, who wants to build a DCS, submits functional and non-functional requirements (QoS) to the System Integrator/Component Assembler. The System Integrator/Component Assembler, in turn, submits the query to the Query Manager. The QM contacts the DSM to get the list of HHs that belongs to the domain specified in the search criteria. Upon receiving this list, the QM creates a mobile agent and sends it across the network to select appropriate components that match the search criteria from each of the HH that is specified in the list. The mobile agent randomly picks one HH at a time, moves to the location of that HH and contacts it locally to get the list of components that matches the search criteria specified in the query. The mobile agent repeats this process until it visits all the HHs specified in the list and returns the component information to the QM. Upon receiving the information, the QM returns the list of appropriate components to the system integrator/component assembler and removes the mobile agent from the system.

With the introduction of mobile agents for the component discovery process of the MURDS, the number of messages that flow across the network reduces because of local execution of tasks as discussed in the following sub-section.

3.1.1 Flow of number of messages between HHs and ARs in the component discovery process

Figure 3.1 shows the flow of messages between a single Headhunter (HH) and a single Active Registry (AR) after the Headhunter multicasts its presence in the discovery process of URDS.
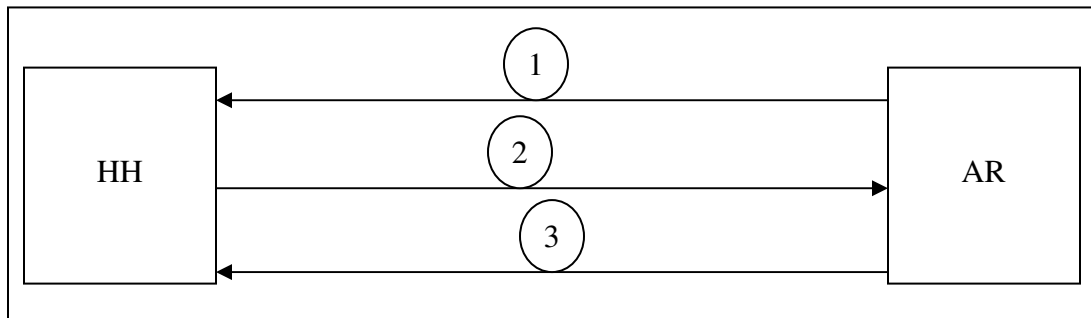
Figure 3.1 Message communication between a HH and an AR in the URDS

(1)   represents the message from the AR to the HH. The AR, which is listening for the multicast messages from HHs at a particular group address, responds to HH's multicast messages by passing its contact information to the HH.

(2) represents the message from the HH to the AR. The HH queries the AR that responds to its announcement for the UniFrame specification of all the components registered with it.

(3)   represents the message from the AR to the HH. The AR responds by passing to the HH the list of components registered with it and the detailed UniFrame specifications of these components.

Therefore, if there exists one HH and one AR that belong to the same multicast group in the discovery process of URDS, it takes three messages to discover components from the AR by the HH. -------- (1)

If there exists one HH and 'N' ARs that belong to the same multicast group in the discovery process of URDS, it takes '3*N' messages to discover components from 'N' ARs by the HH. -------- (2)

If there exists 'M' HHs and 'N' ARs that belong to the same multicast group in the discovery process of URDS, it takes '3*N*M' messages to discover components from 'N' ARs by 'M' HHs. -------- (3)

Figure 3.2 shows the flow of messages between a single HH and 'N' ARs in the discovery process of MURDS
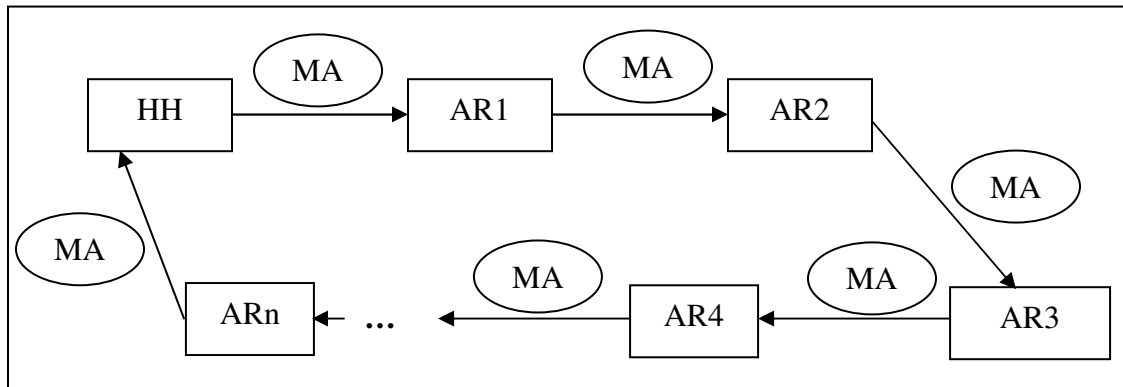
Figure 3.2 Mobile agent based message communication between a HH and 'N' ARs

MA represents a mobile agent that travels on behalf of a HH to gather component information from all the ARs that are specified in the list. The arrow indicates the direction in which the mobile agent travels from one entity to another entity.

If there exists one HH and one AR that belong to the same domain, it takes two messages (i.e., Mobile agent moving from its location to the location of AR is considered as one message. Mobile agent moving back to its original location after completing its

task is considered as one message) to discover components from the AR by the HH with the help of a mobile agent. -------- (4)

If there exists one HH and 'N' ARs that belong to the same domain, it takes 'N+1' messages to discover components from 'N' ARs by the HH with the help of a mobile agent. -------- (5)

If there exist 'M' HHs and 'N' ARs that belongs to the same domain, each HH sends one mobile agent to discover components from 'N' ARs and therefore it takes 'M*(N+1)' messages. -------- (6)

From the above discussion, it is clear that the component discovery process of URDS requires 3*N*M messages, where as the discovery process of MURDS requires M*(N+1) messages. This implies that the discovery process of URDS requires M(2N-1) more messages than the discovery process of MURDS. As N tends to infinity and M tends to infinity, the mechanisms used in MURDS and URDS exhibit O (NM) complexity. This may not make much difference in a situation where the available network resources are abundant but it may matter where the available network resources are less. Since the QMs and the HHs make asynchronous calls to retrieve information from their corresponding peers, they require network connection only during the transfer of mobile agents from one entity to another entity.

3.1.2 Heterogeneous policies

In the URDS, a HH requesting software components will be provided information about all the components that are available with ARs. An important fact that needs to be considered is to limit the availability of services provided by ARs to HHs based upon the type of HHs requesting the services. Many factors, such as security and cost may control the nature of services offered by ARs to the HHs. One factor that is considered in the context of the MURDS is the cost associated with obtaining information from the ARs by the HHs.

A hypothetical scenario based on cost is described below:

In Figure 3.3, AR1, AR2 …….. ARn are 'N' Active Registries from which Headhunters, HH1, HH2 …. HHm, can request information about components. The maintenance of components published in ARs would involve cost for the ARs. In order to recover the costs incurred on them, the ARs may decide to distribute this cost on to the HHs. Thus, the ARs may offer differentiated services, such as: Regular or Premium: based on the nature of service, Individual or Business: based on the usage of service. The nature of services (e.g., a selective release of component information) offered for a **Regular Individual** HH will be different than that offered to a **Premium Business** HH.
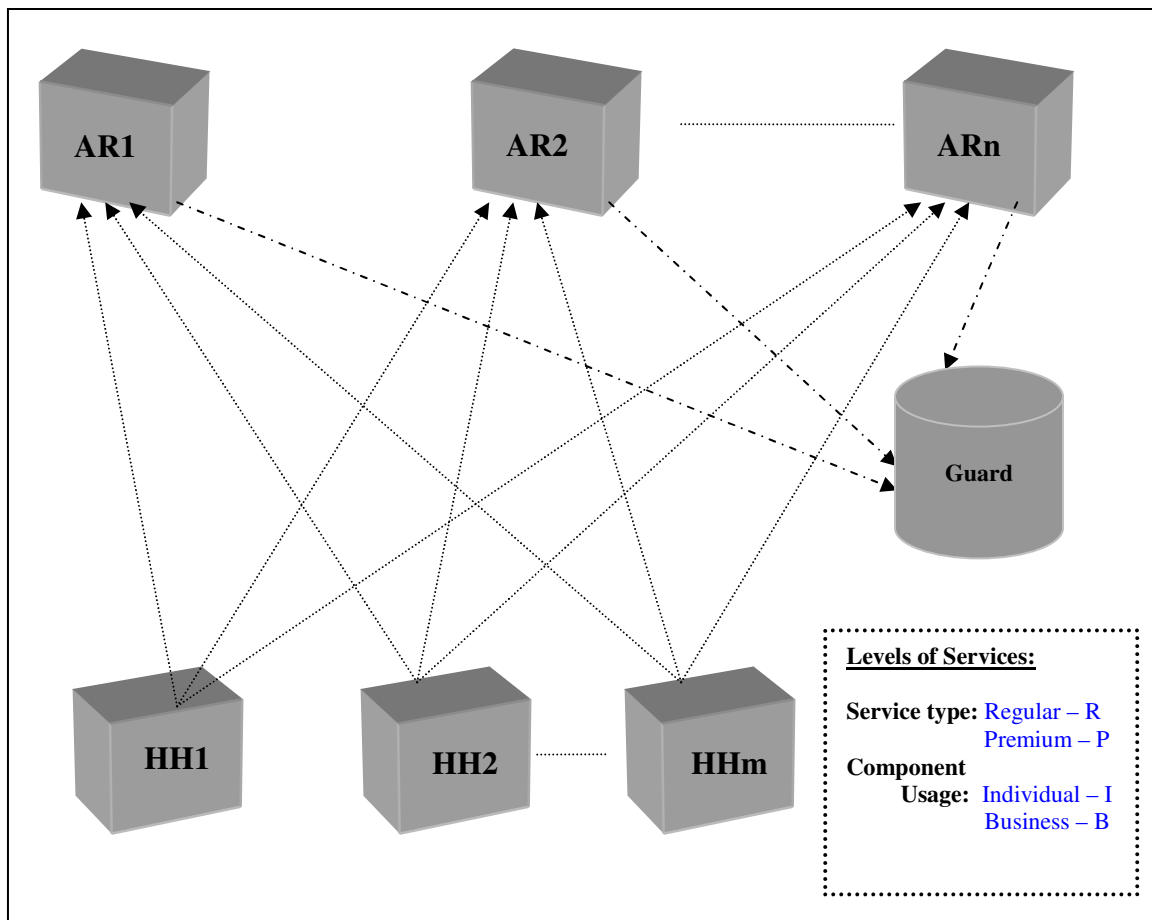
Figure 3.3 Policy relationships between Headhunters and Active Registries

The level of service offered to a HH by an AR will be dependent on many factors, such as the credentials of the HH, the policy employed by the AR and levels of services available. Thus, each HH (or an agent authorized by that HH) will provide its credentials to the AR, before requesting a service, and the AR will offer an appropriate level of service based on its internal decision making process – dependent on its policies and levels of services. It is obvious to expect that different ARs may follow different decision-making processes, thereby, giving rise to heterogeneity. Sending mobile agents on behalf of the HHs is an effective way to discover components from the ARs because of the following reasons:

- Mobile agents would retrieve component information from the ARs by providing appropriate credential information to the ARs.

- As the factors that control decision making of ARs vary from time to time, the policies associated with the HHs also changes. In such scenarios, all the Headhunters have to do is update the level of service associated with different ARs in their repositories and provides the updated information to the mobile agents acting on behalf of them.

One possible characteristic that can demonstrate this heterogeneity and the usage of mobile agents in MURDS is the access-control (AC), which is discussed below:

3.1.2.1 Access Control as an example

The AC-PIM [19] is a generic access control model that can be used to design access control model for any distributed application. The access control model given in [19] can be customized to specific applications to perform access control checks.

According to [19], the main purpose of AC-PIM is:

- To identify the access decision and /or access control elements that perform identical tasks in different component models,

- To unify the vocabulary used to identify these elements in different models, and

- To propose a platform independent model that includes commonly identified and standardized access control and /or access decision elements.

During the application modeling phase and /or the development phase, the AC-PIM enables transformations to the access control platform specific models (AC-PSM) that incorporate access control points. Thus, the AC-PIM provides a clear architectural separation between the access policy (the management and expression of access rules), the access decision (evaluating policy at a given point in time) and the access control (the enforcement of access decisions).

In order to have a controlled access to resources (i.e., components), the following elements are identified for parameterization of the AC-PIM for MURDS:

- *Entities that should pass through access control checks:* Before providing component information to a mobile agent acting on behalf of a HH, the AR needs to know whether it should provide information to the mobile agent or not. Also, it needs to know the level of information that it should give to the mobile agent. Therefore, access control checks are to be made on mobile agents to give access to the information available with the AR.

- *Points within the MURDS architecture where the access control checks should be made*: The access control checks should be made before mobile agents contact ARs. In the AC-PIM [19], the access control check points are guarded by Guards. A Guard is responsible for ensuring that the mobile agent is acting on behalf of a valid HH and is authorized to access the information available with the ARs. The Guard is also responsible for identifying and notifying the level of information that the AR should provide to the mobile agent. The DSM acts as a Guard in the MURDS architecture. The DSM establishes the relationship between each AR and each set of controlling parameters according to the policies of each AR. Conversely, the DSM also establishes a relationship, where in, each HH functions for a definite set of controlling parameters. Figure 3.3 shows the relationship between ARs and the Guard.

- *Application specific context/ attribute information:* The application specific context/ attribute information that a mobile agent needs to specify at the point of an access control check is the information about the HH on whose behalf it is seeking information. It is of the form < headhunter location, mobile agent user name>. Ex: <//192.168.0.100:5001/Headhunter1, HH1Agent>.

In addition to the identification of the above mentioned modeling elements, an access policy is to be defined to perform access control checks on mobile agents. When mobile agents request information about components registered with an AR, the DSM makes use of this access policy and informs the AR about the category of information that the AR should provide to the mobile agent.

In the MURDS, the access control policy is defined as follows:

A mobile agent that acts on behalf of a Headhunter would get component information only if the credentials (i.e., < headhunter location, mobile agent user name>) provided by the mobile agent are authenticated by the DSM.

Any mobile agent that tries to access ARs must submit credentials about the HH to the DSM for authentication and authorization purposes. The DSM consists of a repository of policies associated with valid ARs and HHs that would participate in the component discovery and component selection aspects of the MURDS. Upon initialization, each principal (i.e., the AR or the HH) contacts the DSM with their authentication credentials (i.e., username and password). The DSM authenticates the contacted principal by checking the credentials against the data available in its repository. Also, the DSM registers the credentials in its repository to perform access control checks during the component discovery phase. A mobile agent acting on behalf of a HH carries authentication information with it to get access to the AR. When the mobile agent contacts AR for information, the DSM performs checks on the information provided by the mobile agent and informs the AR about the level of service that it should give to the mobile agent.

3.2 Security Issues

The mobile agent paradigm also has disadvantages. Security issues surrounding the mobile agent paradigm is the main disadvantage that needs to be addressed in the context of MURDS. Mobile agents travel in open distributed environments to carry out specified tasks on behalf of the Headhunters and the Query Managers. While in transit, mobile agents encounter a number of security threats such as confidentiality attacks,

integrity attacks and authentication risks from network entities. Hence, these issues are to be addressed in the context of the MURDS.

o Confidentiality attack: It is defined as "violation of the privacy of a mobile agent due to illegal access or disposal of mobile agent resources by the host environment" [18]. When mobile agents travel from one node to another node, the information that they carry should not be of any potential use to anybody else other than the communicating partners. Since mobile agents travel in an open network channel, it is possible that any interested host can gain an access to the data carried by mobile agents by means of eavesdropping or by masquerading as a legitimate member of the system. In order to avoid these attacks, mobile agents must travel over a secure channel, but it is difficult to physically secure the channel by the communicating partners in an open network. Therefore, mobile agents must be encrypted while in transit so that nobody else other than the communicating partners knows how to decrypt mobile agents.

o Integrity attack: It is defined as "violation of the integrity of a mobile agent due to tampering of agent's code, state or data" [18]. When mobile agents travel in an open network channel, it is possible that they can be modified by means of transmission errors or intentional acts of vandalism. Therefore, a receiving host must be able to identify if the mobile agent is modified or corrupted so that the receiving host can try to reconstruct the agent or ask the sending host to repeat the transmission.

o Authentication risk: It is defined as "jeopardizing the intended goal of a mobile agent by providing false identity" [18]. In today's networked world, it is much easier for imposters to pretend to be a legitimate member of the system under consideration. Hence, they can send mobile agents to or receive mobile agents from legitimate members of the system. Also, they can replay the intercepted data to legitimate members of the system at a later point of time. In order to avoid these authentication risks, all the members that are participating in a communication session must be aware of the real identity of their communication peer by means of authentication checks.

To prevent the above mentioned security attacks on mobile agents, the MURDS system makes use of X.509 [24] certificates and the Secure Socket layer (SSL) [24] protocol. SSL is an industry standard protocol that makes use of both symmetric key cryptography and asymmetric key cryptography to provide confidentiality, data integrity and mutual authentication of sender and receiver.

In the context of the MURDS system, SSL provides confidentiality, data integrity and authentication as follows:

- o SSL provides confidentiality of mobile agents by encrypting them with a symmetric key algorithm that is negotiated using a handshake prior to the actual SSL session and then sending them over a secure socket. Even though mobile agents can still be intercepted by any potential intruder, encryption renders them to be useless.

- o SSL preserves the integrity of the data carried by mobile agents using Message Authentication Codes (MACs). The sending host transmits a mobile agent by attaching a MAC to it, which is calculated on the data carried by the mobile agent using a hash function. The receiving host verifies the integrity of the data carried by the mobile agent by calculating a MAC on the data and comparing it with the MAC attached to the mobile agent. If both MACs are same, the transmitted data is not modified during transmission. Otherwise, the receiving host tries to reconstruct the mobile agent or asks the sending host to retransmit the mobile agent.

- o SSL verifies authenticity of communicating partners during the SSL handshake phase by letting them exchange each others authentication credentials to make sure that each of them are who they are meant to be. Both the communicating partners exchange their personal data and their public keys in the form of X.509 certificates. The combination of certificate and correct private key of each of the communicating partner implies that they are valid entities in the MURDS system.

3.3 <u>Architecture of MURDS</u>

The MURDS architecture follows the guidelines specified for the URDS architecture. The MURDS architecture is organized as a federated hierarchy of ICBs and Headhunters in order to achieve scalability. Each ICB that is participating in the MURDS consists of zero or more HHs attached to it and all ICBs are linked to one another to form a federated group. The HH, which is responsible for discovering services, sends a mobile agent on its behalf to discover services. The QM, which handles requests to find services that match client requirements, sends a mobile agent to retrieve services from the HHs. Both the HH and the QM uses asynchronous mode of communication to achieve their task.

The MURDS architecture handles failure of the HHs and the ARs through periodic announcements and information caching. The DSM maintains a cache of the HHs and a cache of the ARs that are participating in the MURDS. The HHs and the ARs periodically update their availability with the DSM. The DSM periodically checks the duration of the time interval between successive updations of a particular entity and removes it from the system if the time interval exceeds the specified time period.

The MURDS architecture comprises of the following entities to carry out their specified tasks. a) Internet Component Broker (ICB) b) Headhunters (HHs), c) Meta-Repositories, d) Active Registries, e) Services (S1..Sn), and f) Adapter components (AC1...ACn). The following subsections give a description of all the entities specified in the MURDS infrastructure. Figure 3.4 shows interaction of these components in the MURDS architecture.

- *Internet Component Broker (ICB)*: The ICB is not a single entity but a collection of the following services – *Query Manager (QM), the Domain Security Manager (DSM), Link Manager (LM),* and *Adapter Manager (AM).* The ICB acts as an all-pervasive component broker in an interconnected environment. It constitutes the communication infrastructure necessary to identify and locate services, enforce domain security and handle mediation between heterogeneous components. All of the services provided by an ICB are accessible at well-known addresses. It is expected

that there will be a fixed number of ICBs deployed at well-known locations hosted by corporations or organizations supporting UniFrame.

- o *Domain Security Manager (DSM)*: The DSM serves as an authorized third party that handles secret key generation and distribution and enforces group memberships. It also performs access control checks on HHs on behalf of the ARs. In order to perform access control checks, the DSM has a repository of valid users (i.e., HHs, agents acting on behalf of HHs, and the ARs), and the policies associated between valid users (i.e., the ARs and the HHs).

- o *Query Manager (QM)*: The QM is responsible to propagate the component selection criteria that it receives from a system integrator/component assembler to the 'appropriate' HHs. The QM achieves this task by sending a mobile agent on its behalf to select a list of service provider components that match the search criteria. 'Appropriate' HHs are selected based on the domain specified in the requirements. The QM and the LM are responsible for propagating the queries to other linked ICBs.

- o *Link Manager (LM)*: The LM is responsible to establish links between ICBs to form a federation and to propagate queries received from the QM to the linked ICBs. An ICB administrator configures the LM with the location information of other ICBs with which links are to be established.

- o *Adapter Manager (AM)*: The AM acts as a registry/lookup service for clients seeking adapter components. The adapter components register with the AM by specifying the component models that they can bridge efficiently. Clients contact AM to search for adapter components that match their requirements.

- *Headhunters (HHs)*: The HHS are responsible to detect the presence of service providers with the help of mobile agents, to register the functionality of these service providers and to return a list of service providers to the ICB that matches the requirements of the component assembler's/system integrator's request forwarded by the QM.

- *Meta-Repository (MR)*: The MR is a database that is associated with a HH to store the UniFrame specification information of exporters adhering to heterogeneous component models.

- *Active Registry (AR)*: The AR serves as a native registry/lookup service of a particular distributed computing model such as RMI, CORBA, .NET, etc. ARs provide component information to the HHs based on the policies associated with the HHs. ARs have introspection capabilities to discover not only the instances, but also the specifications of the components registered with them.

- *Services (S1…Sn)*: The services that are deployed on the network may be implemented in different distributed component models such as RMI, CORBA, .NET, etc. Each of these services identify themselves by the service type name and the XML description of the component's informal UMM specification.

- *Adapter Components (AC1…ACn)*: The ACs are responsible to serve as bridges between components developed in different distributed component models.

- *Users (C1…Cn)*: The users of the MURDS system can be Component Assemblers, System Integrators/System developers searching for services matching certain functional and non-functional requirements.
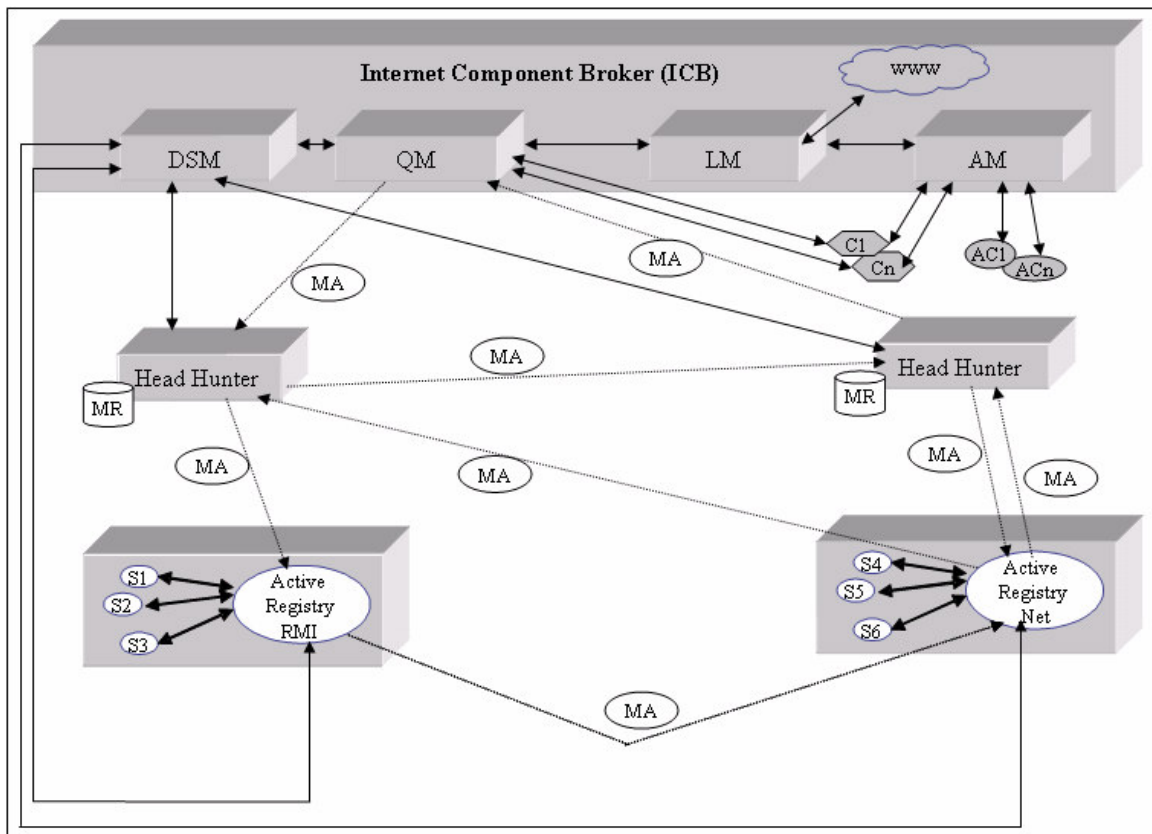
Figure 3.4 MURDS architecture

With the introduction of mobile agents in the MURDS architecture, the functionality of the DSM, the HH, the QM and the AR varies as compared with that of the URDS architecture. Table 3.1 provides significant differences of the functionality provided by the DSM, the HH, the QM and the AR in the MURDS and the URDS architectures. The following subsections provide the high level design details and algorithms for DSM, HH, AR and QM.

Entity	MURDS	URDS
DomainSecurityManager	Serves as an authorized third party that handles secret key generation and distribution and enforces group memberships. In addition, it performs access control checks on HHs on behalf of the ARs.	Serves as an authorized third party that handles the secret key generation and distribution and enforces group memberships and access control to multicast resources through authentication and use of access control lists.
Headhunter	Gets the list of Active Registries from the DSM and uses mobile agent-based communication to discover components from the Active Registries.	Uses multicast messages to identify the Active Registries available in the network and then uses request-reply protocol to discover components from the Active Registries.
QueryManager	Uses mobile agent-based communication to select components from the Headhunters.	Uses request-reply protocol to select components from the Headhunters.
Active Registry	Provides information about	Responds with unicast

	components registered with it based on the policies associated with the Headhunters.	messages to those Headhunters that send multicast messages. Upon receiving a request from the Headhunters for component information, the AR provides the information of all the components registered with it.

Table 3.1 Functional differences of the entities participating in the MURDS and the URDS

3.3.1 Domain Security Manager (DSM)

The DSM is responsible to protect the ARs from unauthorized accesses by mobile agents during the component discovery phase. Therefore, it acts as a Guard and performs the following tasks:

- In order to serve as a Guard, the DSM needs to know all the entities participating in the component discovery phase of the MURDS. Therefore, it maintains a list of authorized users and their passwords as persistent data in the DSM_Repository. Upon initialization, the HHs and the ARs contact the DSM with their authentication credentials. The DSM authenticates and authorizes the contacted entities by checking their authentication credentials with the data available in the DSM_Repository.

- During the component discovery phase, mobile agents originating from HHs submit policy credentials associated with the HHs to the DSM to get access to the ARs. The DSM checks the authenticity of credentials and then authorizes mobile agents to access ARs. Also, the DSM specifies the level of service (i.e., a PremiumBusiness, PremiumIndividual) that the AR must offer to the HH so that the AR can provide appropriate component information to the mobile agents.

- During the component selection phase, the QM contacts the DSM about the list of HHs belonging to a particular domain. The DSM returns the list of appropriate HHs by searching the list of HHs actively participating in the MURDS.

The following sub-sections provide algorithms for DSM functions.

3.3.1.1 Algorithm for DSM initialization

The DSM configuration process involves setting up the DSM_Repository with the information about the domains, authorized users and their passwords, mobile agents associated with users and their passwords. The DSM stores the information in the DSM_Repository as a collection of tables. The DSM configuration is carried out by a system administrator. Upon initialization, the DSM activates the authentication service and thereafter responds to authentication calls from the HHs, the ARs and the mobile agents associated with the HHs.

```
DSM_INITIALIZATION
      CREATE DSM_REPOSITORY
      ACTIVATE DSM_AUTHENTICATION_SERVICE
END_DSM_INITIALIZATION
```

3.3.1.2 Algorithm for authenticating entities

This algorithm outlines the process of authenticating HHs and ARs by verifying their authentication credentials against the DSM_Repository.

```
DSM_AUTHENTICATION_SERVICE
INPUT: userType, userName, password, contactLocation, domain
OUTPUT: boolean value indicating whether the contacted principal is a valid entity
WHILE TRUE
      IF contacted by user
```

isUserAuthenticated =

VALIDATE *userType, userName, password* against

DSM_Repository.

IF *isUserAuthenticated* is TRUE

/* If the user is of type Headhunter, save the information in its

associated table. */

IF *userType* EQUALS "Headhunter"

/* Maintain a list of Headhunter contact locations with their

associated domains in the   *registeredHHTable* table.

*/

Store <hhC*ontactLocation, domainName>* in

registeredHHTable

/* Maintain a list of mobile agents associated with the

authenticated headhunters in a table. */

Store <hhC*ontactLocation, mobileAgentUserName>* in

registeredHHAgentTable

/* If the user is of type active registry, save the information in its

associated tables. */

ELSE IF (*userType* EQUALS "Registry")

/* Maintain a list of active registry contact locations and

their associated domains in a table. */

Store <arC*ontactLocation, domainName>* in

registeredARTable

ENDIF //user type

ENDIF //authorized user

ENDIF //contacted by user

Send a response to the contacted principal.

ENDWHILE

END_DSM_AUTHENTICATION_SERVICE

3.3.1.3 Algorithm to withdraw Headhunters from DSM

This algorithm outlines the process of withdrawing a Headhunter from the DSM.

DSM_WITHDRAW
    INPUT: h*eadhunterLocation*
    REMOVE h*eadhunterLocation entry* from *registeredHHTable*
    REMOVE h*eadhunterLocation entry* from *registeredHHAgentTable*
END_DSM_WITHDRAW

3.3.1.4 Algorithm to respond to a Headhunter's request for list of active registries

This algorithm outlines the process of selecting a list of active registries that belong to the same domain as the Headhunter that requested the list.

DSM_FOR_THE_LIST_OF_ACTIVE_REGISTRIES
    INPUT: *Domain, headhunterLocation*
    OUTPUT: *registryList*
    /* The DSM checks whether the Headhunter is a valid entity to get the AR list. If
    so, then the DSM sends the registryList to the HH. The DSM returns an empty list
    if it cannot find any ARs that belong to the Domain. It can also send an empty list
    if the HH is not a valid entity to get the list of ARs.*/
    IF *registeredHHTable* CONTAINS an entry for *headhunterLocation*
            GET the list of active registries that belong to the *Domain* from the
            availableARTable
            WHILE *availableARTable* CONTAINS active registry entries
                    /* Take one entry at a time from the availableARTable and check
                    if the domain of this entry is same as that of the *Domain*. If so,
                    save the active registry in the registryList table. Repeat this process
                    until all the entities in the list are checked */
                    Key = GET an entry from *availableARTable*
                    IF the domain value of the entry *Key* EQUALS *Domain*

Store < *Key* > in the *registryList*

ENDIF

ENDWHILE

ENDIF

Send *registryList* to the respective Headhunter

END_DSM_FOR_THE_LIST_OF_ACTIVE_REGISTRIES

3.3.1.5 <u>Algorithm to respond to a Query Manager's request for list of Headhunters</u>

This algorithm outlines the process of selecting a list of Headhunters that belong to the domain specified by the QueryManager.

DSM_FOR_THE_LIST_OF_HEADHUNTERS

INPUT: *Domain*

OUTPUT: *headhunterList*

/* The DSM checks for the availability of Headhunters, which belong to the

Domain, in the availableHHTable and returns the list to the QM . IF the DSM

could not find any entires, it returns an empty list to the QM.*/

WHILE *availableHHTable* CONTAINS Headhunter entries

/* Take one entry at a time from the registeredHHTable and check

if the domain of this entry is same as that of the *Domain*. If so, save the

Headhunter location in the headhunterList table. */

Key = GET an entry from *availableHHTable*

IF the domain value of the entry *Key* EQUALS *Domain*

Store < *Key* > in the *headhunterList*

ENDIF

ENDWHILE

Return *headhunterList* to the respective QueryManager

END_DSM_FOR_THE_LIST_OF_HEADHUNTERS

3.3.1.6 <u>Algorithm to validate a mobile agent originating from a Headhunter</u>

This algorithm outlines the process of validating a mobile agent acting on behalf of a Headhunter for component discovery from active registries.

DSM_VALIDATE_HH_MOBILE_AGENT

    INPUT: *headhunterLocation, mobileAgentUserName*

    OUTPUT: *accessLevel*

    /* The DSM checks for the availability of Headhunters, which belong to the Domain, in the availableHHTable and returns the list to the QM . IF the DSM could not find any entries, it returns an empty list to the QM.*/

    IF *availableHHTable* CONTAINS an entry for *headhunterLocation*

        Key = GET the entry from *availableHHTable*

        IF the value of the *Key* EQUALS *mobileAgentUserName*

            //randomly select access level

            accessLevel = GENERATE a random access level

        ENDIF

    ELSE

        SET *accessLevel* to NULL

    END IF

    return *accessLevel* to the respective active registry

END_ DSM_VALIDATE_HH_MOBILE_AGENT

3.3.1.7 <u>Algorithm to update the availability of Headhunters and Active Registries</u>

The Headhunters and the Active Registries periodically contact the DSM to notify their availability in the MURDS system. This algorithm outlines the process of updating the information about the Headhunters and the Active Registries state of availability with the DSM.

DSM_NOTIFY_USER_STATE

    INPUT: userType, user*Location,Domain*

/* If the contacted entity is a Headhunter, register the time stamp of the message received in the registeredHHTimestampTable*/

IF *userType* EQUALS "Headhunter"

    IF *registeredHHTable* CONTAINS an entry for *userLocation* that matches the *Domain*

        Store <user*Location, timeStamp>* in *registeredHHTimestampTable*

    END IF

ELSE IF (*userType* EQUALS "Registry")

    IF *registeredARTable* CONTAINS an entry for *userLocation* that matches the *Domain*

        Store <user*Location, timeStamp>* in *registeredARTimestampTable*

    END IF

    ENDIF //user type

END_DSM_NOTIFY_USER_STATE

3.3.1.8 <u>Algorithm to detect failure of Headhunters and Active Registries</u>

The failure detection algorithm of the DSM involves keeping track of Headhunters and Active Registries which may no longer be alive. The DSM uses the time stamp information available with it and uses this information to purge all those entities that have not responded with in the expected time period.

DSM_UPDATE_LISTS

    WHILE TRUE

        SLEEP TPpurge

        Tc = COMPUTE CURRENT_TIMESTAMP

        WHILE *registeredARTimestampTable* HAS MORE ENTRIES of Active Registries

            Key = GET NEXT ENTRY from *registeredARTimestampTable*

Tr = GET timestamp value of *Key*

IF (Tc-Tr)>2TPpurge

REMOVE *Key* from availableARTable

ENDIF

END WHILE

WHILE *registeredHHTimestampTable* HAS MORE ENTRIES of

Headhunters

Key = GET NEXT ENTRY from *registeredHHTimestampTable*

Tr = GET timestamp value of *Key*

IF (Tc-Tr)>2TPpurge

REMOVE *Key* from availableHHTable

ENDIF

END WHILE

END WHILE

END_DSM_UPDATE_LISTS

3.3.2   Headhunter (HH)

The HH performs the following tasks:

- Detects the presence of service provider components with the help of mobile agents,

- Registers the functionality of discovered components in its local meta-repository, and

- Selects and returns a list of appropriate service provider components to the mobile agent that contacts the HH on behalf of a QM.

The discovery of service provider components is carried out by a mobile agent on behalf of a HH. Once deployed in the UniFrame environment, the HH periodically contacts the DSM to get a list of ARs that belong to the same domain as the HH. Then the HH sends a mobile agent on its behalf to gather component information from all the ARs specified in the list. After visiting all the ARs specified in the list, the mobile agent returns component information including the UniFrame specification of all components to the HH. The HH stores this information in its meta-repository and uses this

information during the match making process to find services that satisfy the computational, co-operational, auxiliary attributes and QoS metrics specified in a search query. The HH periodically contacts the DSM to notify its availability in the discovery process. The following sub-sections provide algorithms for HH functions.

3.3.2.1 Algorithm for HH initialization

At the startup of the MURDS system, the system administrator configures the HH by passing the DSM location information, domain name, username and password as input parameters. Upon initialization, the Headhunter contacts the DSM with its authentication credentials in order to get authorization to participate in the MURDS. Then the Headhunter creates the meta-repository, contacts the DSM to get the list of active registries and sends a mobile agent on its behalf to gather component information from all the active registries that are specified in the list.

HH_INITIALIZATION

 INPUT: *DSMLocation, userType, username, password, domain*
 /* Contact DSM by sending the authentication credentials (userType, username, password, domain, HHLocation) and wait for authorization from DSM. DSM returns a boolean value to indicate the Headhunter's validity in the MURDS. */
 boolean *isUserAuthenticated* = CALL DSM_AUTHENTICATION_SERVICE *with userType, username, password, domain, HHLocation*
 IF *isUserAuthenticated* EQUALS TRUE
  CALL DSM_TO_GET_MOBILE_AGENT_INFO
  CREATE META_REPOSITORY
  CALL HH_PERIODIC_NOTIFICATION
  CALL DSM_FOR_THE_LIST_OF_ACTIVE_REGISTRIES
  CREATE MOBILE_AGENT
  SEND MOBILE_AGENT_TO_DISCOVER_COMP_INFO
 ENDIF
END HH_INITIALIZATION

3.3.2.2 <u>Algorithm to send mobile agent on behalf of a Headhunter</u>

This algorithm outlines the process of discovering service provider components from active registries by a mobile agent.

MOBILE_AGENT_TO_DISCOVER_COMP_INFO

    INPUT: *registryList, registryPolicyList, headhunterLocation*

    OUTPUT: *componentTable*

    WHILE *registryList* HAS MORE ELEMENTS

        /* Mobile agent randomly picks one active registry at a time from the list, moves to the location of active registry and contacts it locally. */

        registryLocation = GET NEXT ELEMENT from *registryList*

        registryPolicy = GET associated policy from the *registryPolicyList*

        MOVE_TO registryLocation

        /*Get the state of the ActiveRegistry*/

        s*tate* = GET the *state* of ActiveRegistry located at registryLocation

        /* IF the Active Registry is in state 2 …randomly select the attribute type*/

        attributeType = “”

        attributeValue = “”

        attribute = randomly select a number between 1 and 3

        IF *attribute == 1*

            attributeType = “algorithm”

            attributeValue = “JFC”

        IF attribute == 2

            attributeType = “compelxity”

            attributeValue = “O(1)”

        IF *attribute == 3*

            attributeType = “technology”

            attributeValue = “Java RMI”

/* Mobile agent contacts active registry to get the component information and stores the received information in the *componentTable. */

componentTable = *CALL* AR_GET_COMPONENT_DATA on active registry with *registryPolicy, attributeType* and *attributeValue* as input parameters

ENDWHILE

/*After visiting all the active registries, return the component information to the headhunter.*/

CALL HH_POPULATE_META_REPOSITORY

END_MOBILE_AGENT_TO_GATHER_COMP_INFO

3.3.2.3 Algorithm for populating Meta Repository

This algorithm outlines the process of populating a headhunter's meta-repository with the information received from a mobile agent [9].

HH_POPULATE_META_REPOSITORY

INPUT: *componentTable*

/* Get one entry from the componentTable at a time and store it in the meta-repository. */

WHILE *componentTable* HAS MORE ELEMENTS

componentInfo = GET NEXT ELEMENT from *componentTable*

STORE *componentInfo* to META_REPOSITORY

ENDWHILE

END_HH_POPULATE_META_REPOSITORY

3.3.2.4 Algorithm to Retrieve Search Results from the Meta-Repository

This algorithm outlines the process in which the Headhunter generates the SQL query from the *queryEntity* and executes this query against the meta-repository to retrieve the list of components matching the search criteria.

HH_EXECUTE_QUERY

     INPUT: *queryEntity*

     OUTPUT: *resultTable*

     sqlQuery = CALL *QM_GENERATE_SQL_QUERY* on *queryEntity*

     resultTable = EXECUTE QUERY *sqlQuery* on *META_REPOSITORY*

     RETURN *resultTable*

END_HH_EXECUTE_QUERY

3.3.2.5 Algorithm to notify DSM about the Headhunter's availability in the system

This algorithm outlines the process of periodically updating a Headhunter's availability in the system.

HH_PERIODIC_NOTIFICATION

     INPUT: *headhunterLocation, Domain*

     CALL DSM_NOTIFY_USER_STATE

END_ HH_PERIODIC_NOTIFICATION

3.3.2.6 Algorithm for Headhunter Shutdown

The Headhunter before shutdown withdraws from the DSM, leaves the multicast group and terminates all the active processes [9].

HH_SHUTDOWN

     // Withdraw Headhunter registration from DSM

     CALL *DSM_WITHDRAW* on *DSM* with *headhunterLocation* as input parameter

HH_SHUTDOWN

3.3.3   Meta-Repository

The Meta-Repository is a repository of information about service provider components adhering to different component models. The Meta-Repository stores the Meta level service information of each component discovered by Headhunter during the discovery process. It comprises of:

• Service type name,

• Details of its informal specification, and

• Zero or more QoS values for the service offered by each of the components.

The implementation of a Meta-Repository is database oriented because of inbuilt search techniques provided by the database to search for components that match the search criteria specified in the query. The Meta-Repository associated with a headhunter is passive in nature because of the fact that the Headhunter brings information and stores it in the Meta-Repository.

3.3.4 Active Registry (AR)

The Active Registries are the native registries that belong to different distributed computing models such as RMI, CORBA and .NET.   The functionality of native registries are extended in such a way that they not only maintain a list of component URLs of the components registered with them, but also maintain detailed UniFrame specifications of components registered with them. The registries use principles of introspection to obtain the URL of XML based specifications of all the components registered with them. The registries parse the specification and maintain the details in a memory resident table. When a mobile agent acting on behalf of a Headhunter contacts the registries for component specifications, they return a restricted amount of information to the Headhunter because of the reasons specified under section 3.2. The following sub-sections provide algorithms for AR functions:

3.3.4.1 Algorithm for AR Initialization

At the startup of the MURDS system, the system administrator configures the AR by passing the DSM location information, domain name, username and password as input parameters. Upon initialization, the AR contacts the DSM with its authentication credentials in order to get authorization to participate in the MURDS. Then the AR starts processing requests that it receives from mobile agents acting on behalf of the Headhunters.

AR_INITIALIZATION

      INPUT: *DSMLocation, userType, username, password, domain*

      /* Contact DSM by sending the authentication credentials (userType, username, password, domain, ARLocation) and wait for authorization from DSM. DSM returns a boolean value to indicate the validity of active registry in the MURDS. */

      boolean *isUserAuthenticated* = CALL DSM_AUTHENTICATION_SERVICE on *dsm with userType, username, password, domain, HHLocation*

      IF *isUserAuthenticated* EQUALS TRUE

          AR_PERIODIC_NOTIFICATION

          ACTIVATE AR_GET_COMPONENT_DATA

      ENDIF

END_AR_INITIALIZATION

3.3.4.2 <u>Algorithm for obtaining UniFrame Specifications of Registered Components</u>

This algorithm [9] outlines the process for obtaining the UniFrame specifications of the components registered with an AR. The AR gets a URL list of all the components registered with it. It then steps through this list and gets a handle to each of these components. Using the component reference, the AR examines the component's properties to check for a property returning the URL of its UniFrame specification. The AR then reads the XML-based UniFrame specification from the URL and parses this specification to obtain all the component details, which it stores in an entity object

componentEntity. The AR builds a hash table of such entities corresponding to each of the components registered with it and returns this hash table to the Headhunter.

AR_GET_COMPONENT_DATA

INPUT: *headhunterLocation, mobileAgentUserName, attributeType, attributeValue*

OUTPUT: *componentTable*

WHILE TRUE

    IF contacted by a mobile agent to retrieve component data

        serviceType = DSM_VALIDATE_HH_MOBILE_AGENT

        IF *serviceType* NOT EQUALS " "

            CREATE a new *componentTable*

            //Obtain a list of object URL's of all objects registered with

            //this registry.

            registeredServicesURLList = GET LIST of service

            components registered with this Active Registry

            IF *serviceType* EQUALS PB

                accessLevel = ALL

            ELSE IF *serviceType* EQUALS PI

                accessLevel = L1L2

            ELSE IF *serviceType* EQUALS RB

                accessLevel = L1

            ELSE IF *serviceType* EQUALS RI

                accessLevel = L2

            // For each object in this URL list

            FOR   i=0   to   LENGTH   of   *registeredServicesURLList*

            registeredServiceURL = *registeredServicesURLList[i]*

            // Lookup and obtain the reference to the services from the

            // registry using the registered service URL.

            serviceObject = LOOKUP *registeredServiceURL*

            // Obtain the location (URL) of the UniFrame Specification

            // for this service by introspecting its property name called

// "uniFrameSpecification".

uniFrameSpecURL = CALL *AR_INTROSPECT_PROPERTY* with

serviceObject, "uniFrameSpecification"

// Parse the UniFrame Specification and construct a

// *componentEntity* which can be persisted.

CREATE a *componentEntity*

document = PARSE *uri* and load XML document

CALL *AR_PARSE_UNIFRAME_SPEC* with *document*

//Add the component to the *componentTable*

cost = GET *cost* of *componentEntity*

IF *attributeType* EQUALS "" AND *attributeValue* EQUALS ""

    IF *accessLevel* EQUALS ALL

        PUT < *registeredServiceURL, componentEntity>*

        in *componentTable*

    ELSE IF *accessLevel* EQUALS L1L2

        IF *cost* EQUALS L1

            PUT < *registeredServiceURL,*

            componentEntity> in *level1ObjectTable*

        ELSE IF *cost* EQUALS L2

            PUT < *registeredServiceURL,*

            componentEntity> in *level2ObjectTable*

    ELSE IF cost EQUALS accessLevel

            PUT < *registeredServiceURL,*

            componentEntity> in *componentTable*

ELSE IF *attributeType* EQUALS "algorithm"

    algorithms [] = GET *algorithms* of *componentEntity*

    algorithmValue = CHECK for *attributeValue* in

    algorithms []

    IF *accessLevel* EQUALS ALL AND *algorithmValue*

    EQUALS TRUE

PUT < *registeredServiceURL, componentEntity>*

 in *componentTable*

ELSE   IF   *accessLevel*   EQUALS   L1L2   AND *algorithmValue* EQUALS TRUE

IF *cost* EQUALS L1

PUT   <   *registeredServiceURL, componentEntity>* in *level1ObjectTable*

ELSE IF *cost* EQUALS L2

PUT   <   *registeredServiceURL, componentEntity>* in *level2ObjectTable*

ELSE IF cost EQUALS accessLevel AND *algorithmValue* EQUALS TRUE

PUT < *registeredServiceURL, componentEntity>* in *componentTable*

ELSE IF *attributeType* EQUALS "complexity"

complexity = GET *complexity* of *componentEntity*

IF *accessLevel* EQUALS ALL AND *complexity* EQUALS *attributeValue*

PUT < *registeredServiceURL, componentEntity>*

 in *componentTable*

ELSE IF *accessLevel* EQUALS L1L2 AND *complexity* EQUALS *attributeValue*

IF *cost* EQUALS L1

PUT   <   *registeredServiceURL, componentEntity>* in *level1ObjectTable*

ELSE IF *cost* EQUALS L2

PUT   <   *registeredServiceURL, componentEntity>* in *level2ObjectTable*

ELSE IF cost EQUALS accessLevel AND *complexity* EQUALS *attributeValue*

PUT < *registeredServiceURL, componentEntity>* in *componentTable*

ELSE IF *attributeType* EQUALS "technology"

technologies *[]* = GET *technologies* of *componentEntity*

technologyValue = CHECK for *attributeValue* in *technologies []*

IF *accessLevel* EQUALS ALL AND *technologyValue* EQUALS TRUE

PUT < *registeredServiceURL, componentEntity>* in *componentTable*

ELSE IF *accessLevel* EQUALS L1L2 AND *technologyValue* EQUALS TRUE

IF *cost* EQUALS L1

PUT < *registeredServiceURL, componentEntity>* in *level1ObjectTable*

ELSE IF *cost* EQUALS L2

PUT < *registeredServiceURL, componentEntity>* in *level2ObjectTable*

ELSE IF cost EQUALS accessLevel AND *technologyValue* EQUALS TRUE

PUT < *registeredServiceURL, componentEntity>* in *componentTable*

ENDFOR

IF accessLevel EQUALS L1L2

size = GET number of elements in *level1ObjectTable*

IF *size* IS GREATER THAN *ZERO*

WHILE *i* LESS THAN OR EQUAL TO *size*/2

GET *componentEntity* from *level1ObjectTable*

PUT < *registeredServiceURL, componentEntity>* in *componentTable*

i++

```
                    ENDWHILE
            ENDIF
            size = GET number of elements in level2ObjectTable
            IF size IS GREATER THAN ZERO
                    WHILE i LESS THAN OR EQUAL TO size/2
                            GET componentEntity from level2ObjectTable
                            PUT < registeredServiceURL, componentEntity> in
                            componentTable
                            i++
                    ENDWHILE
            ENDIF
        RETURN componentTable
    ENDIF
ENDWHILE
END_AR_GET_COMPONENT_DATA
```

3.3.4.3 Algorithm for Parsing the UniFrame Specification

This algorithm [9] uses recursion to parse through the nodes of the XML tree, extract the node values and store these values in the *componentEntity*. The algorithm starts parsing at the root node element. It extracts the node name and checks if the node name matches any attribute in *componentEntity* and populates it with the value in this node. It then finds all the children of that node and repeats the process through recursion.

```
AR_PARSE_UNIFRAME_SPEC
    INPUT: nodeElement
    nodeName = GET NODE NAME from nodeElement
    FOR each attribute in componentEntity
        IF nodeName EQUALS attribute
        nodeValue = GET NODE VALUE from nodeElement
            SET attribute value in componentEntity to nodeValue
```

       ENDIF

ENDFOR

// Get the list of children for this Node.

NODELIST *childrenList* = GET CHILDNODES for *nodeElement*

IF *childrenList* NOT NULL

       // For every child node in the list

       FOR i = 0 to LENGTH of *childrenList*

              childNode = childrenList[i]

              // Recurse through the function READ_NODE passing it the

              // childNode as reference.

              CALL *AR_PARSE_UNIFRAME_SPEC* with *childNode*

       ENDFOR

ENDIF

END_AR_PARSE_UNIFRAME_SPEC

3.3.4.4 Algorithm for Introspection of the Registered Components

This algorithm [9] outlines the process for examining a service object to find a specific property and retrieve its value. The algorithm gets a list of all the properties from the service object and tries to find a match for the specific property of interest. Once the property is found, a handle to the Read accessor method (getter) of this property is obtained and invoked.

AR_INTROSPECT_PROPERTY

       INPUT: *serviceObject, propertyName*

       OUTPUT: *property*

       //Introspect and retrieve all information pertaining to this service object.

       serviceObjectInfo = INTROSPECT *serviceObject* to retrieve object information

       //Get a description list of all properties of this object.

       propertyDescriptorList =

       GET PROPERTY DESCRIPTORS from *serviceObjectInfo*

//Iterate through the description list to find the desired property.

FOR i=0 to LENGTH of *propertyDescriptorList*

 propertyDescriptor = propertyDescriptorList[i]

 propertyDescriptorName = GET NAME of *propertyDescriptor*

 //If the property descriptor name matches the desired property name

 IF *propertyDescriptorName* EQUALS *propertyName*

  *//*Get the accessor method that returns the property value.

  method = GET READ METHOD of *propertyDescriptor*

  //Invoke the method to retrieve the value.

  property = INVOKE *method* of *serviceObject*

  //Return this property to the requester.

  RETURN *property*

 ENDIF

ENDFOR

END_AR_INTROSPECT_PROPERTY

3.3.4.5 Algorithm to notify DSM about the Active Registry's availability in the system

This algorithm outlines the process of periodically updating an AR's availability in the system.

AR_PERIODIC_NOTIFICATION

 INPUT: *registryLocation, Domain*

 CALL DSM_NOTIFY_USER_STATE

END_ AR_PERIODIC_NOTIFICATION

3.3.5 Query Manager (QM)

The QM is responsible for finding services matching the client's query request. The QM parses the user's request into a structured query language statement and sends a mobile agent to find service components from the list of 'appropriate' Headhunters. Appropriate Headhunters are determined based on the domain specified in the client query. After completing the job of finding service components from the list of Headhunters, the mobile agent returns information to the QM. The QM, in turn, returns the information to the client.

Selection of results by the QM is controlled by the client's parameter values in the query. The user would be required to enter the following details:

- Service details such as domain, name, description, and function;
- The functional attributes such as algorithms, complexity, and technology
- Search by auxiliary attributes such as mobility, security, fault tolerance
- Search by QoS parameters such as end-to-end delay and availability

Each entered parameters would be considered as a constraint to the query. As more parameters are entered, the query becomes more constrained, whereas a more global query would have fewer parameter values entered.

The QM queries are handled in the following manner:

- Parse the client's entered parameters and extract the text pertaining to the various UniFrame specified attributes necessary for finding service components.
- Compose the extracted information into a SQL based query statement.
- Contact the DSM and get the list of Headhunters belonging to the domain specified in the client query request.
- Create a mobile agent, submit the query to the mobile agent and delegate the job of selecting appropriate service components from the list of Headhunters.
- The mobile agent contacts each Headhunter specified in the list and requests the component information. The Headhunter searches its local Meta-Repository with the query submitted by the mobile agent and returns the list of service components. After

visiting all the Headhunters, the mobile agent returns the service component information to the QM.

- The QM returns the results, whether there are results or not, to the client of the system.

The following sub-sections provide algorithms for AR functions:

3.3.5.1 Algorithm for QM Initialization

The QM initialization activates the client request handler. This process receives requests from clients and responds with results.

QM_INITIALIZATION
      ACTIVATE *QM_CLIENT_REQUEST_HANDLER*
END_QM_INITIALIZATION

3.3.5.2 Algorithm for handling query requests from clients

This algorithm outlines the process for servicing requests from clients.

QM_CLIENT_REQUEST_HANDLER
INPUT: *naturalLanguageQuery*
OUTPUT: *resultTable*
WHILE TRUE
      IF contacted by client with *naturalLanguageQuery* Request
            headhunterList=CALL DSM_FOR_THE_LIST_OF_HEADHUNTERS
            IF headhunterList IS NOT EMPTY
                  queryEntity = Parse *naturalLanguageQuery*
                  sqlQuery = QM_GENERATE_SQL_QUERY
                  CREATE MOBILE_AGENT
                  SEND MOBILE_AGENT_TO_SEARCH_FOR_COMP_INFO

WHILE requestResultsFound is FALSE

//keep waiting for results

ENDWHILE

ENDIF

ENDIF//contacted by client

SEND resultTable to client

ENDWHILE

QM_CLIENT_REQUEST_HANDLER

3.3.5.3 Algorithm to send mobile agent on behalf of a QM

This algorithm outlines the process of selecting service components from Headhunters by a mobile agent.

MOBILE_AGENT_TO_SEARCH_FOR_COMP_INFO

INPUT: *sqlQuery, headhunterList, queryManagerLocation*

OUTPUT: *resultTable*

WHILE *headhunterList* HAS MORE ELEMENTS

/* Mobile agent randomly picks one Headhunter at a time from the list, moves to the location of that Headhunter and contacts it locally/

headhunterLocation = GET NEXT ELEMENT from *headhunterList*

MOVE_TO headhunterLocation

/* Mobile agent contacts Headhunter to get the component information based on the sqlQuery and stores the received information in the *resultTable. */

resultTable = CALL HH_EXECUTE_QUERY on Headhunter

ENDWHILE

/*After visiting all the Headhunters, return the component information to the Query Manager.*/

CALL QM_RECEIVE_RESULTS

END_MOBILE_AGENT_TO_SEARCH_FOR_COMP_INFO

3.3.5.4 Algorithm for Generating Structured Query Language (SQL) Statement

This algorithm forms a part of the operations performed by *queryEntity*. This algorithm outlines the process for generating a SQL statement based on the attributes extracted from the client's query. These attributes are captured in the *queryEntity*. The attribute names, values and constraints stored in the query entity are built into a SQL statement.

QM_GENERATE_SQL_QUERY

      INPUT: *queryEntity*

      OUTPUT: *sqlQuery*

      attributeList = GET all attributes in *queryEntity*

      FOR i=0 to LENGTH of *attributeList*

            attribute = *attributeList[i]*

            IF *attribute* is selected as search parameter

                  attributeValue = GET value of this attribute from *queryEntity*

                  attributeConstraint = GET constraint value from *queryEntity*

                  CONCATENATE to BUILD SQL query with the

                  attribute, *attributeConstraint*, and *attributeValue*

            ENDIF

      ENDFOR

      RETURN *sqlQuery*

END_QM_GENERATE_SQL_QUERY

3.3.5.5 Algorithm to send query results to the QM by the mobile agent

This algorithm outlines the process of mobile agent returning component selection results to the QM.

QM_RECEIVE_RESULTS

INPUT: *resultTable*

SET requestResultsFound to TRUE

END_HH_POPULATE_META_REPOSITORY

3.3.6 <u>Services</u>

Service Exporter Components belong to different distributed component models, e.g., Java RMI, CORBA, EJB, etc. The components are identified by their *Service Offers* comprising a) service type name, b) informal UniFrame specification, and c) zero or more QoS values for that service. A component registers its interfaces with an Active Registry. The component interface contains a method that returns the URL of its informal specification. The informal specification is stored as an XML file adhering to certain syntactic contracts to facilitate parsing. These service exporter components will be tailored for specific domains, such as Financial Services, Health Care Services, Manufacturing Services, and will adhere to the relevant standards, business architectures, and research and technologies for these industry specific markets.

This chapter provided the design details for DSM, HH, AR and QM. Chapter 4 presents a prototypical implementation for the MURDS architecture.

4. MURDS IMPLEMENTATION

A prototype for the MURDS architecture is implemented by enhancing the prototypical implementation of the URDS architecture [9]. The significant differences between the MURDS architecture and the URDS architecture are the introduction of mobile agents instead of request-reply type of communication for component discovery and component selection phases and the introduction of heterogeneous policies. Therefore, a part of the functionality that is identical in both the MURDS and the URDS architectures has been reused for the MURDS prototype implementation and the functionality for mobile agents has been implemented in the MURDS. The following sub-section gives an overview of the technology used for the prototype implementation of the MURDS architecture.

4.1 Technology

The prototype implementation of the MURDS is based on the architectural model laid out by [48] in J2EE. J2EE defines a standard that applies to all aspects of architecting, developing, and deploying multi-tier, server-based applications. Figure 4.1 [from 49] shows the components of the J2EE Model.

The J2EE platform specifies technologies to support multi-tier enterprise applications. These technologies are classified into three categories [48]: Component, Service, and Communication. The technologies that have been used from these categories in the MURDS implementation are described below:

• Component Technologies

Components are application level software entities. The J2EE Component technologies have been used in the MURDS prototype to develop the front-end client components and back-end service components. All J2EE component technologies depend on the runtime support of a system-level entity called a "Container" that provides components with services such as life cycle management, security, deployment and threading.
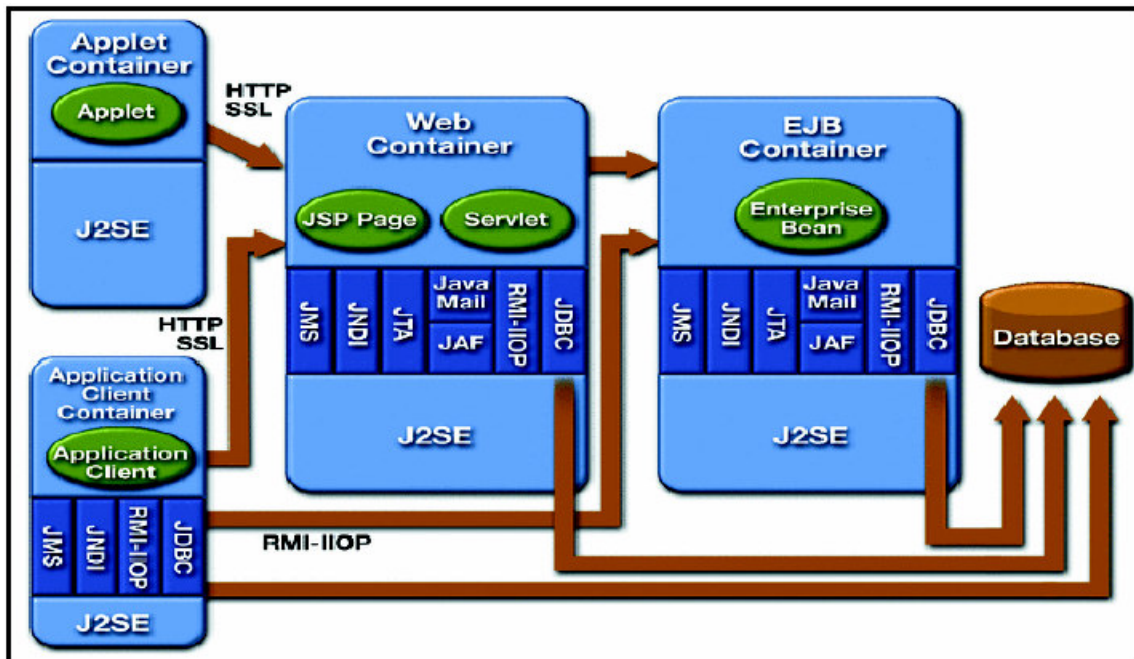
Figure 4.1 Components and Containers of J2EE Model. (Figure 4.1 is courtesy of SUN Microsystems, [48])

The MURDS prototype consists of *Application Clients*. *Application clients* are client components that execute in their own Java Virtual machine and are hosted in an *Application Client Container*. Application clients are implemented as Java RMI components.

- Service Technologies

    The J2EE platform service technologies allow applications to access a variety of services. The prominent service technologies supported are *JDBC^{TM} API 2.0* which provides access to databases, *Java Transaction API (JTA) 1.0* for transaction processing, *Java Naming and Directory Interface (JNDI) 1.2* which provides access to naming and directory services, and *J2EE Connector Architecture 1.0* which supports access to enterprise information systems.

The service technologies used in the prototype are described below:

- *JDBC^{TM} API 2.0:* The JDBC^{TM} API provides methods to invoke SQL commands from Java programming language methods. The JDBC API has two parts: an application-level interface used by the application components to access a database, and a service provider interface to attach a JDBC driver to the J2EE platform.

Java API for XML Processing 1.1: XML is a language for representing text-based data so the data can be read and handled by any program or tool. Programs and tools can generate XML documents that other programs and tools can read and handle. Java API for XML Processing (JAXP) supports processing of XML documents using DOM, SAX, and XSLT parsers. JAXP enables applications to parse and transform XML documents independent of a particular XML processing implementation.

- Communication Technologies

Communication technologies provide mechanisms for communication between clients and servers and between collaborating objects hosted by different servers. Some of the communications technologies supported by the J2EE Platform include – Transport Control Protocol over Internet Protocol (TCP/IP), Hypertext Transfer Protocol HTTP 1.0, Secure Socket Layer SSL 3.0, Java Remote Method Protocol (JRMP), Java IDL, Remote Method Invocation over Internet Inter ORB Protocol (RMI-IIOP), Java Message Service 1.0 (JMS), JavaMail and Java Activation Framework.

The prototype uses Java Remote Method Invocation to achieve inter-component communication between the following components: the DSM and the QM, the DSM and the HH, the DSM and the AR.

In addition to the above mentioned technologies, the MURDS prototype requires a mobile agent platform to create mobile agents and to send them across the network during the component discovery and component selection phases. The MURDS prototype uses Grasshopper [40] - a Java-based mobile agent platform for this purpose. The communication service provided in the Grasshopper allows transport of agents between the following components with the help of Java RMI connections: HH and the AR , the

QM and the HH, and the HHs. The Grasshopper uses Secure Socket Layer SSL for secure transportation of mobile agents from one host to another host.

4.2 Prototype Implementation

This section describes an implementation of a prototype for the MURDS architecture. Figure 4.2 illustrates this implementation. The MURDS prototype implementation follows multi-tier architecture, which is generally used for distributed applications. The prototype consists of three tiers namely the client tier, the middle tier and the database tier. Various parts of the prototype implementation falls into these three tiers. The *Client tier* of the prototype supports application clients. The *middle tier* of the prototype supports client services through Java-RMI based component services (i.e. DSM, HH, QM, and AR) and Grasshopper enabled mobile agents. The *Database tier* supports access to the repositories by means of standard APIs.
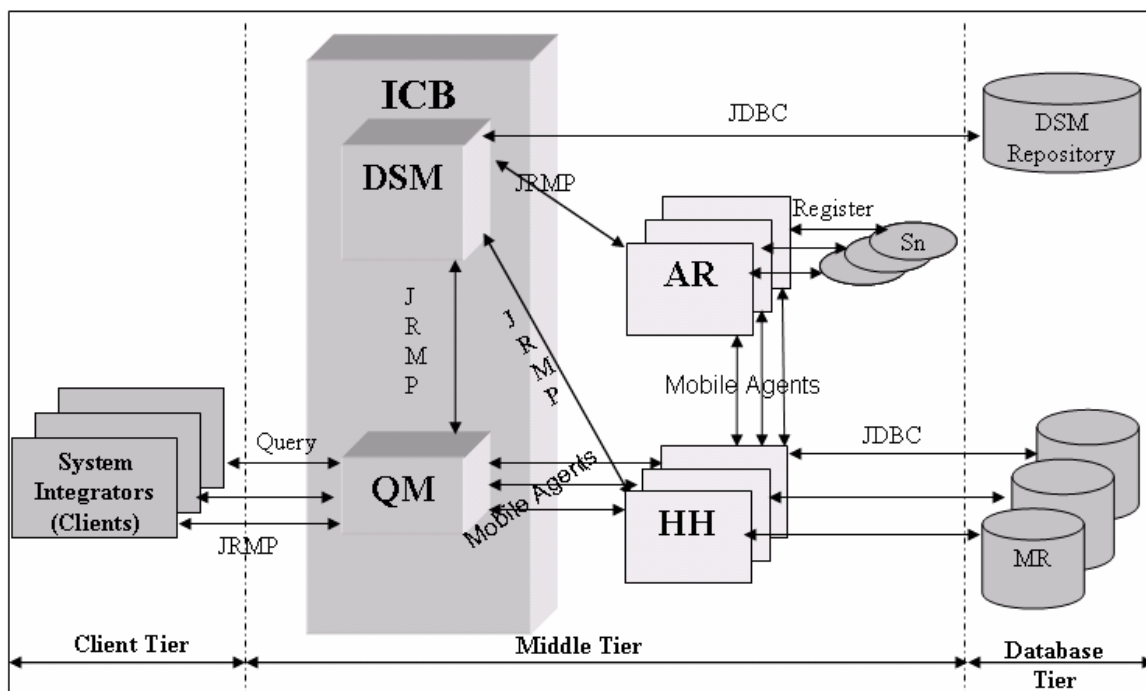
Figure 4.2 MURDS Implementation

4.2.1 Platform and Environment

The Java 2 Platform, Standard Edition (J2SE) [49] version 1.4.2 software environment is used to implement the algorithms outlined for various components of the MURDS prototype. The core architectural components (DSM, QM, HH, and AR) are implemented as Java-RMI based services. The entity that processes client requests is implemented as Java-RMI based service. The Grasshopper version 2.2.4 is used to implement mobile agents. The repositories (DSM_Repository and Meta_Repositories) are implemented as databases on Oracle v 9.2.

4.2.2   Communication Infrastructure

The unicast communication between the core architectural components is achieved through JRMP. The communication service provided by the Grasshopper Distributed Agent Environment is used to transfer mobile agents between the core components. The connections to the Oracle databases are established using JDBC APIs. An interaction between the entity that processes client requests and the QM is achieved through JRMP.

4.2.3   Security Infrastructure

Mobile agents require a secure communication channel to travel on the network. In order to provide a secure communication channel, Grasshopper 2.2.4 uses the Java Secure Socket Extension (JSSE). JSSE enables secure internet communications by implementing a Java version of Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols. Mobile agents are authenticated by checking their credentials against the data available with the DSM.

4.2.4 Programming Model

The prototype implementation of the MURDS is divided into modules and these modules are decomposed into specific objects to represent the behavior and data of the application. All the objects in the MURDS implementation have been categorized as follows: *Entity Objects, Helper Objects, Persistent Data, Component Services* and the *Agent Objects* depending upon their functionality.

4.2.4.1 Entity Objects

Entity objects serve to represent the individual rows in a database as objects or to encapsulate an application specific concept in terms of an object. The entity objects in the prototype support access or methods to set and retrieve the values of the attributes they hold. These objects can be passed by value as serializable Java objects. The prototype contains the following entity objects: Component, *QueryBean.* Class diagrams for these objects are presented in the Appendix A.

Component: The attributes of the *Component* class mirror the fields of the table UMMSpecification. The *Component* has functionality built in to store it to a database.

QueryBean: The *QueryBean* encapsulates the attributes of a Query received from the client. The bean also has the logic associated with generating a SQL query based on the attributes it holds.

4.2.4.2 Helper Objects

The prototype uses Helper objects for purposes such as data access or for performing specific utility functions such as obtaining component attributes from XML files, periodically updating the HH status and the AR status with the DSM. The following classes serve as Helper classes and have been sub-classified under the categories of Data Access Objects and Dependent Objects.

4.2.4.2.1 Data Access Objects

The data access objects used in the prototype encapsulates access to databases. The DomianSecurityManager object and the Headhunter object access their respective databases through the data access objects.

The class diagrams of the data access objects are given under Appendix B. A description of the data access objects used in the prototype is provided here:

DSMRepositoryHelper: This class performs functions associated with accessing the *DSM_Repository* to retrieve user-domain mappings, user-agent mappings and for user authentication.

MetaRepositoryHelper: This class performs functions associated with storing information in the *Meta_Repository* during component discovery process and accessing the *Meta_Repository* to retrieve search results during the component selection process.

SQLEngine: This class acts as a wrapperthat encapsulates the essential JDBC API methods and performs functions associated with establishing database connections and executing the queries. It is used by the *DSMRepositoryHelper* and *MetaRepositoryHelper* classes.

4.2.4.2.2 Dependent Objects

The prototype uses dependent objects for performing utility functions. These dependent objects are immutable and the objects that create and use them manage their life cycle.

The class diagram of the dependent objects is given under Appendix C. A description of the dependent objects is provided below:

UniFrameIntrospector: This utility class uses reflection to analyze the properties of an object and retrieve the value corresponding to a specific attribute.

UniFrameSpecificationParser: This class is used to parse a UniFrame XML specification file and construct an instance of the *ConcreteComponent*.

BuildPersist: This class is used to store a ConcreteComponent in the Meta-Repository during the component discovery process and to build a concrete component during the component selection process.

ARStateRenewer: This class operates as a *Thread* which executes periodically. This class is used to update the availability of an Active Registry service with the DomainSecurityManager Service.

HHStateRenewer: This class operates as a *Thread* which executes periodically. This class is used to update the availability of a Headhunter service with the DomainSecurityManager Service.

DiscoveryAgent: This class operates as a *Thread* which executes periodically. This class is used to send a mobile agent on behalf of the Headhunter service to discover software components from a list of Active Registry Services.

ComponentSelectionAgent: This class is used to send a mobile agent on behalf of the QueryManager service to select software components from a list of Headhunter Services.

PrincipalAvailabilityChecker: This class operates as a *Thread* which executes periodically. This class is used to update the list of Headhunter Services and the list of Active Registry Services available with the DomainSecurityManager Service.

ServerObject: The ServerObject, which is used by HHAgent and the QMAgent, is described below:

In the context of the MURDS implementation, mobile agents are implemented using Grasshopper. These mobile agents consider QM object and AR object as external application objects. Therefore, a service is required which can communicate with the external objects on behalf of mobile agents. The ServerObject enables mobile agent objects (i.e., the HHAgent object and the QMAgent object) to interact with the service objects (i.e., the Query Manager object, the Headhunter object and the Active Registry object).

IServerObject: This is the interface for the ServerObject. This interface publishes the following methods:

- postCompDataToHH: This method is invoked by the HHAgent. The purpose of this method is to return the list of components discovered by the HHAgent on behalf of the Headhunter.

- getCompDataFromAR: This method is invoked by the HHAgent. The purpose of this method is to discover components from the active registry objects on behalf of the Headhunter.

- getCompDataFromHH: This method is invoked by the QMAgent. The purpose of this method is to search for components that are available with the Headhunters by the QMAgent on behalf of the QueryManager.

- postCompDataToClient: This method is invoked by the QMAgent. The purpose of this method is to return the list of components retrieved from the Headhunter objects by the QMAgent on behalf of the QueryManager.

- getARState():This method is invoked by the HHAgent. The purpose of this method is to return the state of the ActiveRegistry to the Headhunter.

ServerObject: This class implements *IServerObject* interface. The ServerObject serves as a communication medium between HHAgent/QMAgent and AR/QM/HH. Grasshopper provides a service called external communication service in order to let mobile agents communicate with those applications that are not implemented completely on Grasshopper platform. The HHAgent and the QMAgent uses the external communication

service to communicate with the ServerObject in order to accomplish their job in the MURDS prototype implementation.

4.2.4.3 Persistent Data

The prototype maintains persistent data that is associated with the DomainSecuirtyManager Service and the Headhunter Service as database tables. The databases that are used in the prototype are the DSM_Repository and the Meta_Repository.

DSM_Repository

The *DSM_Repository* comprises of five tables *Users*, *Permissions*, *User_Permission_Xref*, and *Mobile_Users*. Appendix D provides the schema for the DSM_Repository.

The *Users* table serves to hold the Headhunter/ActiveRegistry/QueryManager information. The columns of this table are *userid* (numeric identifier), *usertype* (whether Headhunter or Registry), *username*, and *password*. The primary-key in this table is the *userid*. An example of a record of this table is as follows: <1,'Headhunter', 'Headhunter1', 'xxxx'>.

The *Permissions* table serves to hold a list of allowed permissions (or domains). The columns in this table are *permissionid* (numeric id for permission) and *permissionname* (domain name like Finance, Manufacturing, etc.,). The *permissionid* serves as the primary-key for this table. An example of a record of this table is as follows: <1, 'Manufacturing'>.

The *User_Permission_Xref* table serves to map the permission to be assigned to a user. The table contains two numeric columns *userid* which references the *userid* in *Users* and *permissionid* which references *permissionid* in *Permissions*. The combination

of the *<userid,permissionid>* serve as the primary key. An example of a record of this table is as follows: <1, 2>.

The *Mobile_Users* table serves to hold mobile agents information associated with the Headhunter/QueryManager. The columns of this table are *mobileuserid* (numeric identifier), *mobileusername*, *mobileuserpassword,* principal*type* (whether Headhunter or querymanager), and principal*name*. The primary-key in this table is the mobile*userid*. An example of a record of this table is as follows: <1,'HH1Agent', 'HH1Agent', 'Headhunter', 'Headhunter1'>.

Meta_Repository

The Meta_Reposiotry comprises of twelve tables UMMSpecification, Algorithms, RequiredInterfaces, ProvidedInterfaces, Technologies, ExpectedResources, DesignPatterns, KnownUsages, Aliases, PreProcessing, PostProcessing, and CompFuncQoS. Appendix E provides the schema for the Meta_Repository.

The *UMMSpecification* table serves to hold QoS parameters of the components discovered during the component discovery phase. The columns in this table are *componentname, subcase*, domianname, systemname, description, id, version, author, creatingdate, *validity, atomicity, registration, model, purpose, complexity, mobility, security, faultolorance, qoslevel, cost,* and *qualitylevel*. The primary-key in this table is the *id*. An example of a record of this table is as follows: <'DeluxeDocumentServer', 'DeluxeDocumentServerCase1', 'Document', 'DocumentManager', 'provide document service to the document terminal', 'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', '1.0', 'zhisheng Huang', 'August 2002', 'Yes', 'Yes', 'magellan.cs.iupui.edu:8500/HeadHunter1', 'provide document services to the document terminal.', 'O(1)', 'No', 'L1', 'L1', 'L1', 'L1', 'L1' >.

The *Algorithms* table holds information about algorithms used in implementing the component. The columns in this table are *id, componentname, domainname, systemname,* and *algorithm*. The combination of the <id, algorithm> serves as the

primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document', 'DocumentManager', 'JFC'>.

The *RequiredInterfaces* table holds information about interfaces that are required for implementing the component. The columns in this table are *id, componentname, domainname, systemname,* and *interface*. The combination of the <id, interface> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document', 'DocumentManager', 'IDocumentManagementCase1'>.

The *ProvidedInterfaces* table holds information about interfaces that are provided for implementing the component. The columns in this table are *id, componentname, domainname, systemname,* and *interface*. The combination of the <id, interface> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document', 'DocumentManager', 'IDocumentManagementCase1'>.

The *Technologies* table holds information about component technology utilized in the implementation of the component. The columns in this table are *id, componentname, domainname, systemname,* and *technology*. The combination of the <id, technology> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document', 'DocumentManager', 'Java RMI'>.

The *ExpectedResources* table holds information about the resources that are required for executing the component. The columns in this table are *id, componentname, domainname, systemname,* and *expectedresource*. The combination of the <id, expectedresource> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document', 'DocumentManager', 'Memory: 1.0Gb'>.

The *DesignPatterns* table holds information about the resources that are required for executing the component. The columns in this table are *id, componentname, domainname, systemname,* and *pattern*. The combination of the <id, pattern> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document' , 'DocumentManager', 'N/A'>.

The *KnownUsages* table holds information about the known usages of the component. The columns in this table are *id, componentname, domainname, systemname,* and *usage*. The combination of the <id, usage> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document', 'DocumentManager', 'N/A'>.

The *Aliases* table holds information about the alias names of the component. The columns in this table are *id, componentname, domainname, systemname,* and *alias*. The combination of the <id, usage> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer',   'Document', 'DocumentManager', 'N/A'>.

The *PreProcessing* table holds information about the dependency of a component on other components. The columns in this table are *id, componentname, domainname, systemname,* and *collaborator*. The combination of the <id, collaborator> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document', 'DocumentManager', 'DocumentTerminalCase1'>.

The *PostProcessing* table holds information about other components that may depend on the discovered component. The columns in this table are *id, componentname, domainname, systemname,* and *collaborator*. The combination of the <id, collaborator>

serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'Document', 'DocumentManager', 'DocumentDatabaseCase1'>.

The columns in the *CompFuncQoS* table are *id, componentname, systemname, qosparameter,* and *value*. The combination of the <id, functionname, qosparameter, value> serves as the primary key. An example of a record of this table is as follows: <'magellan.cs.iupui.edu:9000/DeluxeDocumentServer', 'DeluxeDocumentServer', 'DocumentManager', 'N/A', 'N/A '>.

4.2.4.4 <u>Service Components</u>

A service component here refers to a software unit that provides a service. The service provided could be a computational effort or an access to underlying resources. A service component consists of one or more artifacts (software, hardware, libraries) that are integrated together to provide the service. Service Components are described by a interface and associated implementation. Each Service Component is a stand-alone functional unit, which accepts inputs through its published interfaces, and returns results. Clients or other components can remotely access a service component using standard communication protocols. For a component to be remotely accessible, it should be deployed in a runtime environment (e.g., Application Server) with ports open to receive incoming requests and return results.

The service components implemented in the prototype are the DSM, QM, HH, and AR. These service components utilize the Entity Objects and Helper Objects to achieve their functionalities and store and retrieve information from their respective repositories. Appendix F provides class diagrams for the Service Components.

- Domain Security Manager object

IDomainSecurityManager: This is the interface for the DomainSecurityManager object. This interface publishes the following methods:

- o *getHHListForDomain*: This method is invoked by the QueryManager. The purpose of this method is to return a list of registered Headhunters for a particular domain to the QueryManager.

- o *authenticationService*: This method is invoked by the Headhunter and the ActiveRegistry components to authenticate themselves with the DSM.

- o *getARListForDomain*: This method is invoked by the Headhunter. The purpose of this method is to give the list to a mobile agent to discover components from active registries.

- o *renewARState*: This method is invoked by the active registry to inform its presence in the MURDS.

- o *renewHHState*: This method is invoked by the Headhunter to inform its presence in the MURDS.

- o *receiveUpdatedTables*: This method is invoked by the PrincipalAvailabilityChecker to update the freshness of Headhunters and active registries.

- o *getARTimestampTable*: This method is invoked by the PrincipalAvailabilityChecker to get the time stamps associated with the registered active registries.

- o *getHHTimestampTable*: This method is invoked by the PrincipalAvailabilityChecker to get the time stamps associated with the registered headhunters.

- o *authenticateHHAgent*: This method is invoked by the ServerObject to check the authenticity of agents coming from multiple Headhunters.

- o *getMobileAgentInfo*: This method is used by the Headhunter to get the mobile agent identity that it can use for sending mobile agent to the active registry for component information.

DomainSecurityManager: This class implements *IDomainSecurityManager* interface.

- Headhunter object

IHeadhunter: This is the interface for the *Headhunter* object. This interface publishes the following methods:

- o *performSearch*: This method is invoked by the *QMAgent* to retrieve components that match search criteria specified in the query.
- o populateMetaRepository: This method is invoked by the *ServeObject* to populate Headhunter's Meta_Repository with the components discovered from active registries by the HHAgent.

Headhunter: This class implements *IHeadhunter* interface.

- Active Registry object

IActiveRegistry: This is the interface for the *ActiveRegistry* object. This interface publishes the following method:

- o *getComponentData:* This method is invoked by the ServerObject to retrieve component information from the *ActiveRegistry object*.
- o *getState():* This method is invoked by the ServerObject to retrieve the state from the *ActiveRegistry object*

ActiveRegistry: This class implements the *IActiveRegistry* interface.

- Query Manager object

IQueryManager: This is the interface for the *QueryManager* service. This interface publishes the following method:

- o *getSearchResultTable*: This method is invoked by the *RequestProcessor* to obtain a list of services matching the search criteria specified by a component assembler.

QueryManager: This class implements the *IQueryManager* interface.

4.2.4.5 <u>Mobile Agent Objects</u>

The mobile agent objects that are used by the Headhunter object and the Query Manager object are shown below:

- HHAgent object: An instance of this class act as a mobile agent on behalf of the Headhunter object to discover components from the Active Registry object. This class is implemented using mobile agent API provided by the Grasshopper 2.2.4. It consists of the following methods:
  - *init*: The purpose of this method is to serve as a constructor for the HHAgent. As soon as the HHAgent is created by the Headhunter object, the init method on the HHAgent is automatically invoked by an '*agency*' to initialize class variables with the input parameters passed on by the Headhunter object. This method is called only once during the lifetime of an agent (i.e. during its creation). In Grasshopper, an agency is the agent execution environment for mobile agents.
  - *aftermove*: This method is automatically invoked by the hosting agency after an agent migrates to a new agency. This method is implemented to create a proxy of a ServerObject, which enables HHAgent communication with the Active Registry object.
  - *live*: This method is used to implement the tasks that are to be performed by the HHAgent after moving to a new agency. The new agency invokes this method on the HHAgent after invoking aftermove method.

- QMAgent object: An instance of this class act as a mobile agent on behalf of the QM object to search components from the Headhunter object. This class is implemented using mobile agent API provided by Grasshopper. It consists of the following methods:
  - *init*: The purpose of this method is to serve as a constructor for the QMAgent. As soon as the QMAgent is created by the QM object, the init method on the QMAgent is automatically invoked by an '*agency*' to initialize class variables

with the input parameters passed on by the Query Manager object. This method is called only once during the lifetime of an agent (i.e. during its creation). In Grasshopper, an agency is the agent execution environment for mobile agents.

o *aftermove*: This method is automatically invoked by the hosting agency after an agent migrates to a new agency. This method is implemented to create a proxy of a ServerObject, which enables QMAgent communication with the Headhunter object.

o *live*: This method is used to implement the tasks that are to be performed by the QMAgent after moving to a new agency. The new agency invokes this method on the QMAgent after invoking aftermove method.

This chapter presented the implementation details of the MURDS prototype that enhances the URDS architecture. The prototype of the URDS was implemented in Java and utilized Java RMI for communication between service entities. The MURDS prototype enhanced the URDS prototype by replacing the Java RMI based communication between different service entities (i.e., a HH and an AR , a QM and a HH) with the mobile agent based communication for various reasons. The next chapter presents the details of the experiments performed using this prototype to validate the MURDS architecture proposed in chapter 3.

5. VALIDATION

In order to validate the performance of the prototype, an empirical experimentation was performed. Due to the limited number of Windows-OS systems available for experimentation, fewer Headhunters, Active Registries, Query Managers and clients were used than would be optimal for these experiments. Future work would include the use of more systems to run a large number of Headhunters, Active Registries, Query Managers and clients in order to evaluate the scalability of the MURDS prototype. The testing of the MURDS prototype was carried out by using the components that belong to a Banking system called Super Bank.

The service components that belong to the Super Bank are as follows:

- ATM – a Java RMI component which requires services such as depositMoney, withdrawMoney, transferMoney, checkBalance, validate.

- CustomerValidationServer – a Java RMI component which provides service to validate a ATM client.

- EconomicTransactionServer - a Java RMI component which provides services such as depositMoney, withdrawMoney, transferMoney, checkBalance to a client.

- CashierTerminal – a Java RMI component which requires services such as openAccount, closeAccount, depositMoney, withdrawMoney, transferMoney, checkBalance, validate.

- CashierValidationServer – a Java RMI component which provides service to validate a CashierTerminal client.

- DeluxeTransactionServer - a Java RMI component which provides services such as depositMoney, withdrawMoney, transferMoney, checkBalance to a client.

- TransactionServerManager – a Java RMI component which provides services such as openAccount, closeAccount to a client.

All the experiments for prototype validation were conducted on four PCs running Windows XP Professional. Sun's JDK 1.4.2_04 was used on all the four PCs to run the service objects and the client objects of the MURDS system. Grasshopper 2.2.4 was used

on all the four machines to provide an agent execution environment for agents working on behalf of their corresponding service objects (i.e., the Headhunter and the Query Manager). The Secure Socket Layer communication service provided by the Grasshopper was used to securely transmit mobile agents over the network. Oracle v 9.2 available on the Phoenix server was used to maintain databases associated with the DomainSecurityManager object and the Headhunter object.

5.1 Experimentations

The experiments that were carried out serve the following purposes:

- To validate the proposed MURDS architecture, and
- To show that the system is scalable.

The metrics that were used to carry out different experiments are described below:

- Client Query Result Retrieval Time (CQRRT): CQRRT is defined as the time taken from the point of issue of query by the client to the point of retrieval of results back to the client.

- Message Consumption (MC): MC is defined as the number of messages taken by a mobile agent, which acts on behalf of a Headhunter, to discover components from Active Registries during the Component Discovery Phase.

5.1.1   Experimentation to validate the proposed architecture

The main purpose of the MURDS is to identify and retrieve components that match client search criteria, with the help of mobile agents. For this purpose, the architecture is provided in section 3.3 and the functionality of various entities involved in the architecture is provided under sections 3.3.1 to 3.3.6. One way to validate the functionality of each entity is to perform experiments based on appropriate metrics and analyze the empirical data available for those metrics. According to the functionality, the following things are expected:

- An increase in the number of components that match the search criteria would increase the time taken by a mobile agent to retrieve components from appropriate Headhunters participating in the system.

- Similarly, an increase in the number of Headhunters participating in the system would increase the time taken by a mobile agent to retrieve components from those Headhunters.

- An increase in the number of Active Registries participating in the system would have negligible effect on the time taken by the mobile agent to retrieve components from the Headhunters.

    In order to measure the time for the above mentioned three cases, CQRRT is considered as a metric because it involves the events of discovering components as well as selecting appropriate components based on a client request. If the empirical data obtained by varying the number of components, Headhunters and Active Registries verifies the trend expected for the above three cases, then it can be said that the MURDS system works with varying number of components, Headhunters and Active Registries.

- One of the reasons for introducing mobile agents into the MURDS architecture is to reduce the number of network resources consumed for the component discovery process. In order to verify the network resource consumption, MC is considered as an appropriate metric. If the empirical data obtained by conducting an experiment to measure the number of messages consumed during the discovery process matches the message consumption model presented under section 3.1.1, then it validates that the mobile agent-based communication is advantageous when compared to that of the request-reply communication.

    An important point that needs to be noted is that the nature of messages that take place during the component discovery process of MURDS is different when compoared to that of the URDS. As per the message consumption model given under section 3.1.1, the transfer of a mobile agent from one host to another host is considered as a single message. Similarly, the unicast communication between a Headhunter and an Active Registry is considered as a single message. However, the amount of information carried by a mobile agent during its transfer from one host to

another host increases by a specific amount along the path of its travel. The amount of information transferred through unicast communication is constant between a Headhunter and an ActiveRegistry.

- Another reason for introducing mobile agents into the MURDS architecture is to effectively discover components from Active Registries by providing appropriate credentials to the Active Registries by mobile agents acting on behalf of different Headhunters. The Active Registries exists in different states at different points of time and offer varying sets of data to the mobile agents that request information from them. The mobile agents periodically discover information about newly registered components from different Active Registries and submit the information to their respective Headhunters. The Headhunters assess usefulness of different components available in their local Meta-Repositories for component selection process and intimates the type of components that they require to the mobile agents. The mobile agents sense the state of Active Registries and use its intelligence to request the type of components, which would be useful to the Headhunters, from the Active Registries. Hence, the empirical data obtained by varying the policies associated between Headhunters and Active Registries varies accordingly, then it validates that mobile agents would effectively discover components from ARs with varying policies.

5.1.2 Experimentation to show that the prototype is scalable

Any software application that is distributed in nature should be able to serve its purpose effectively with an increase in the number of entities participating in the system. The MURDS system should be able to serve a client request effectively with an increase in the number of HHs, ARs, components and the client requests. Also, it should be able to process multiple queries sent to the system at a given point of time. The set of experiments with varying number of HHs, ARs and the components described under section 5.1.1 holds good for scalability purposes as well. An experiment based on average CQRRT is to be performed to show that the MURDS system can handle multiple queries at a time. According to the functionality provided, the CQRRT would increase with an increase in the number of simultaneous queries processed by the MURDS system.

Due to the lack of available systems, scalability experiments were conducted on fewer systems with the limited number of Headhunters, Active Registries, and Components. With the addition of more Windows OS systems to the MURDS, future experimentation would include more number of Headhunters, Active Registries, and Components.

5.2 Results

To discuss the effect of increase in the number of Headhunters, the Active Registries, the queries and the Components on CQRRT, the chain of events that occur during the component selection process is given below:

1. When a client issues a query to the Query Manager, the Query Manager contacts the DomainSecurityManager to get the list of Headhunters belonging to the domain specified in the query.
2. The Query Manager creates a mobile agent, specifies the query to the mobile agent and sends it to select appropriate components from the list of Headhunters.
3. The mobile agent randomly selects a Headhunter from the list, moves to the location of that Headhunter and contacts it locally to get the appropriate components. The Headhunter searches its local Meta-Repository and returns appropriate components to the Headhunter.
4. After receiving the results from the Headhunter, the mobile agent repeats step 3 until it visits all the Headhunters specified in the list.
5. Then the mobile agent contacts the Query Manager and returns the results to it.
6. The Query Manager in turn returns the results to the client.

The following sub-sections discuss the results obtained for various experiments. The measurements that were taken for these experiments were averaged over thirty trials.

5.2.1 Increase in the number of Components

The parameters that were used for this experiment are as follows:

- One Query Manager
- One Client

- One Headhunter
- Seven Active Registries
- Number of Components registered with Active Registries was gradually increased from zero to eight.

CQRRT vs. Increase in Number of Components

	1	2	3	4	5	6	7	8	9
CQRRT(ms)	6395.3	8335.8	9964	11762	13862	15228	17595	20231	22344

CQRRT(ms) (y-axis), Number of Components (x-axis)
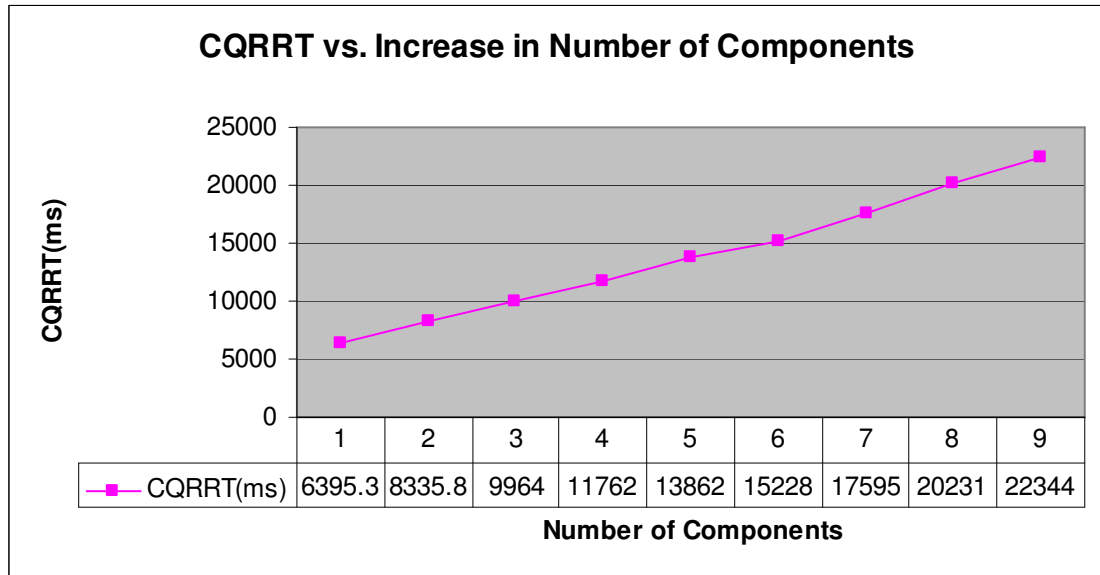
Figure 5.1 CQRRT vs. Number of Components

Figure 5.1 indicates that an increase in the number of components would increase the CQRRT. This trend could be attributed to the fact that an increase in the number of components registered with Active Registries increases the size of the Headhunter's Meta-Repository. When the Headhunter receives a query to find appropriate components, it searches for components in its Meta-Repository. The more the size of Meta-Repository, it would take more time for the Headhunter to retrieve appropriate components. Also, if the retrieved results consist of a greater number of components, the Headhunter would spend more time in building those components by getting the information from the Meta-Repository. Hence, the time taken to retrieve results would in turn increases the CQRRT proportionately assuming that the time taken by the QM to get the list of Headhunters from the DomainSecurityManager and to send the mobile agent is invariant for each query.

5.2.2   Increase in the number of Active Registries

The parameters that were used for this experiment are as follows:

- One Query Manager

- One Client

- One Headhunter

- Four Components were registered with each Active Registry

- Number of Active Registries was gradually increased from one to seven.

CQRRT vs. Number of Active Registries

	1	2	3	4	5	6	7
CQRRT(ms)	13862.4	14075.3	18265.6	18021.8	18403	18353.1	18103.3

Number of Active Registries
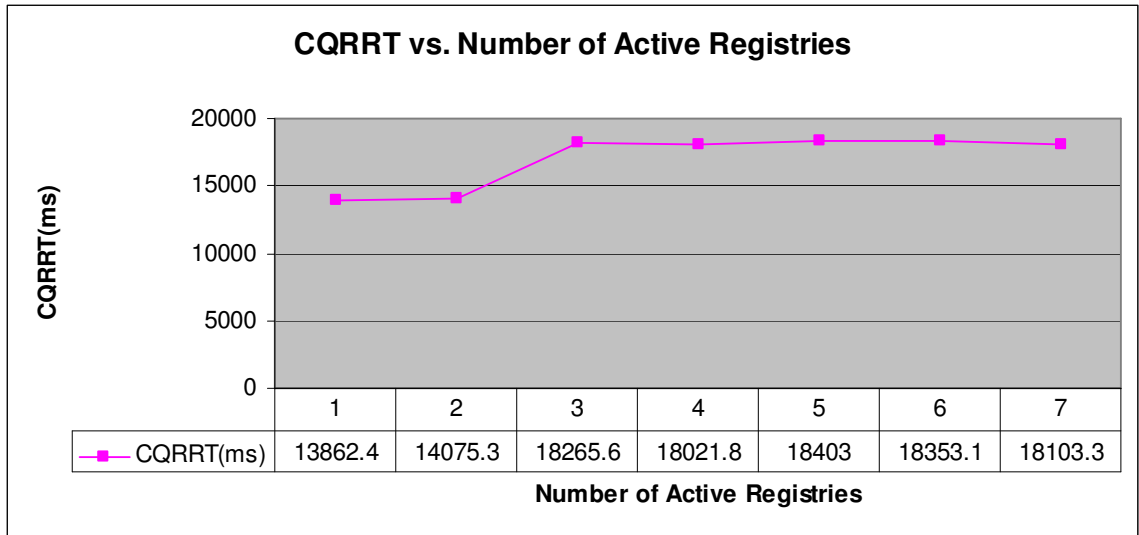
Figure 5.2 CQRRT vs. Number of Active Registries

Figure 5.2 indicates that an increase in the number of Active Registries participating in the discovery process has negligible effect on the CQRRT. The chain of events explained under section 5.2 indicates that the component selection process does not involve Active Registries. The negligible increase in the CQRRT would be attributed to the fact that the Headhunter might have received the query from the mobile agent acting on behalf of a QM while it was updating its local Meta-repository with the information obtained from the component discovery process. This would have delayed the Headhunter from accessing the information from its Meta-Repository.

5.2.3   Increase in the number of Headhunters

The parameters that were used for this experiment are as follows:

- One Query Manager

- One Client
- Seven Active Registries
- Four Components were registered with each Active Registry
- Number of Headhunters was gradually increased from one to five

CQRRT vs. Increase in Number of Headhunters

	1	2	3	4	5
CQRRT(ms) for MURDS	26275.2	45068.6	63684.5	81687.6	99690
CQRRT(ms) for Stationary Agent based URDS	26292.64	54353.73	82665.71	122488.3	195112.8

Number of Headhunters
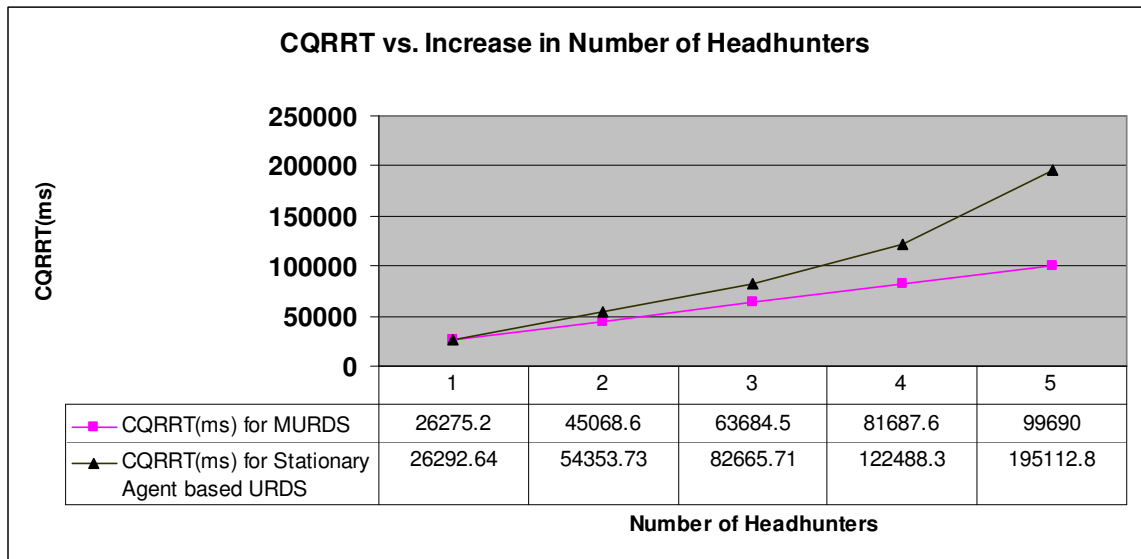
Figure 5.3 CQRRT vs. Number of Headhunters

To compare the performance of mobile agent based resource discovery service with the stationary agent based resource discovery service, experiments were carried out on both the prototypes. The following paragraphs provide an explanation of the component selection process in both the prototypes.

In a mobile agent based resource discovery service, when a QueryManager receives a query from the client, it sends a mobile agent to search components from a list of appropriate Headhunters. The mobile agent randomly picks one Headhunter at a time, moves to the location of that Headhunter and retrieves components that match the query. Then it moves to the next Headhunter in the list and repeats this process till it visits all the Headhunters. Finally, the mobile agent sends the reulst to the QueryManager. The QueryManager and the mobile agent acting on behalf of it communicate with each other through asynchronous mode of communication.

In a stationary agent based resource discovery service, when a QueryManager receives a query from the client, it creates a stationary agent to search components from a

list of appropriate Headhunters. The stationary agent randomly picks one Headhunter as a primary Headhunter (PH) and assigns it the job of selecting components from the remaining list of Headhunters. The PH searches for components in its local MetaRepository. Then it creates a stationary agent and delegates the job of propagating query to the remaining list of Headhunters. The stationary agent randomly picks one Headhunter as a PH from the remaining list of Headhunters and assigns it the job of selecting components from the remaining list of Headhunters. This step is repeated until the query is passed to all the Headhunters in the list. The stationary agent at each step is responsible for sending the results back to the entity (i.e., either a Headhunter or a QueryManager) that creates it. The results are finally returned back to the Client. The QueryManager, the Headhunters and the Stationary agents communicate with one another through synchronous mode of communication.

Figure 5.3 indicates that an increase in the number of Headhunters would increase the CQRRT for both the mobile agent based component selection process and the stationary agent based component selection process. Reason is that the agents in both the processes contact one Headhunter at a time to retrieve component information from that HH. Each Headhunter consumes time to retrieve components from its local Meta-Repository and to return those components to the agents. As the number Headhunters specified in the search list increases, the amount of time that the agents take to return results to the Query Manager increases proportionately. This, in turn, increases the CQRRT. Figure 5.3 indicates that the stationary agent based component selection process consumes more time in retrieving results from a set of Headhunters when compared to that of the mobile based component selection process. The reason for this could be attributed to the differences in the synchronous and asynchronous mode of communication because all the other steps involved in the both the processes are similar in nature.

5.2.4   Increase in the number of Queries

The parameters that were used for this experiment are as follows:
- Three Query Managers
- Three Headhunters

- Seven Active Registries
- Four Components were registered with each Active Registry
- Three Clients
- Number of queries was gradually increased from ten to hundred and twenty. All the queries were equally distributed among three clients.

CQRRT vs Increase in number of Queries

Number of Queries	1	2	3	4	5	6	7	8	9	10	11	12
Average CQRRT(secs)	16.1	28.8	43.7	55.6	67.9	79.6	93.7	113	116	135	150	163
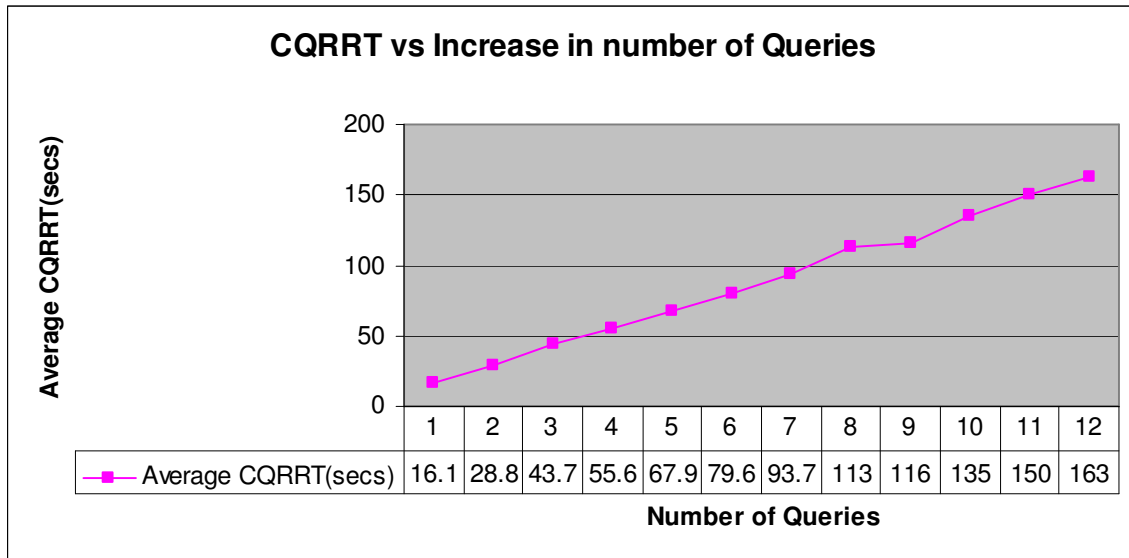
Figure 5.4 CQRRT vs. Number of Queries

Figure 5.4 indicates that an increase in the number of simultaneous queries sent to Query Managers would increase the average CQRRT proportionately. As the number of incoming queries increases, Headhunters become busy in servicing those requests in addition to the requests that they are still servicing; so an increase in the CQRRT will be the result.

Similar experiments were conducted for the URDS. The nature of the graph shown in the above figure was similar to the graph obtained for the URDS.

5.2.5   Message Consumption

The parameters that were used for this experiment are as follows:

- One Headhunter
- Number of Active Registries are gradually increased from one to seven

Number of Active Registries vs. Message Consumption

	1	2	3	4	5	6	7
Message Consumption in the MURDS	2	3	4	5	6	7	8
Message Consumption in the URDS	3	6	9	12	15	18	21

Number of Active Registries
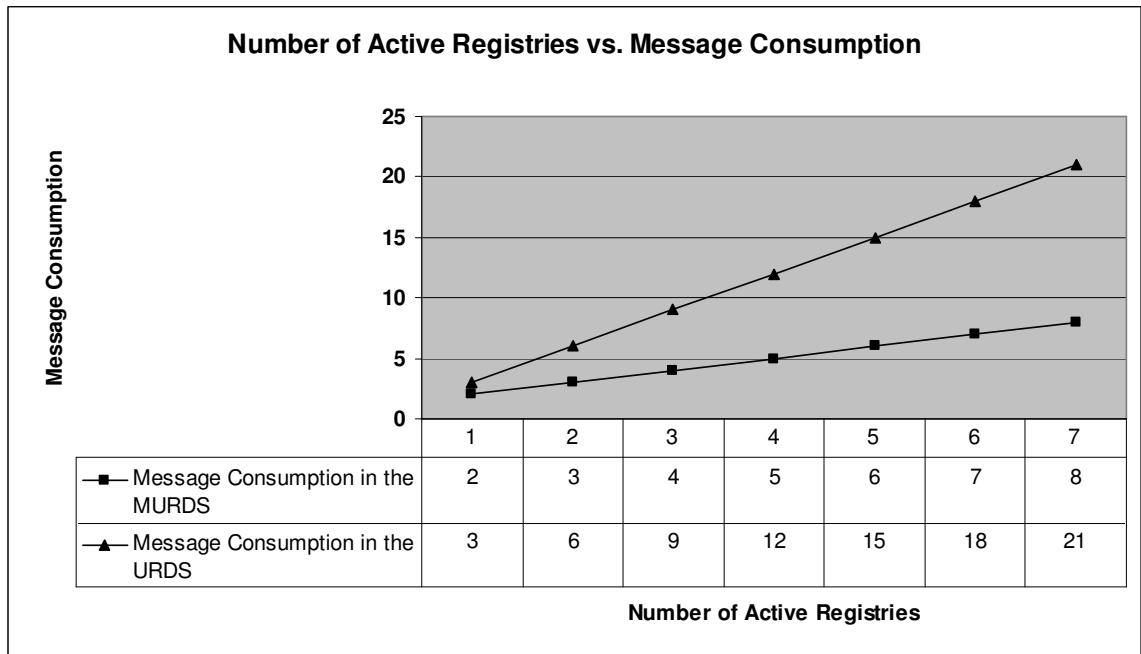
Figure 5.5 Number of Active Registries vs. Message Consumption

This experiment was conducted to see whether the number of messages consumed during the component discovery process matches the message consumption model presented under section 3.1.1. Figure 5.5 indicates that a mobile agent acting on behalf of a Headhunter consumed (N+1) messages to discover components from 'N' Active Registries where as a Headhunter consumed 3N messages to discover components from 'N' Active Registries using request-reply communication. Hence, this experiment validated the message consumption model presented under section 3.3.1.

5.2.6   Heterogeneous policies

This experiment was conducted to verify whether different ARs would provide differentiated services to the mobile agent acting on behalf of a HH based on appropriate credentials presented to the ARs. This experiment was carried out with two ARs and one HH. The services provided by an AR to a HH were based on whether the HH belongs to any one of the following service types: PremiumBusiness or PremiumIndividual or RegularIndividual or RegularBusiness. All the components that belong to Super Bank

were divided into two cost levels (i.e., L1 and L2). The policies associated with all the ARs in the experiment were as follows:

- A premium Business (PB) HH would get all the components that belong to L1 and L2.

- A premium Individual (PI) HH would get a subset of components that belong to both L1 and L2.

- A Regular Business (RB) HH would get all the components that belong to L1.

- A Regular Individual (RI) HH would get all the components that belong to L2.

The HH that was used in the experiment was associated with the ARs with any one the policies mentioned above. All the ARs participating in the discovery process exists in either state1 or state2 at different point of times. When an AR is in state1, it searches for components based on the policies mentioned above and return all those components to the mobile agents. When an AR is in state2, it searches for components based on functional attributes of the UMMSpecification of a component and returns only those components that match the functional attributes.

During the component discovery phase, a mobile agent acting on behalf of a HH enquires about the state of an AR. If an AR is in state1, the mobile agent submits credentials (i.e., < *headhunter location, mobile agent user name*>) to the AR. The AR verifies the credentials with the DSM and returns varying number of components based on the policy associated with the HH. If an AR is in state2, the mobile agent submits different set of credentials (i.e., < *headhunter location, mobile agent user name, functional attribute, functional attribute value*>) when compared to the set of credentials submitted to the AR when it is in state1. The AR verifies the credentials with the DSM, then selects components based on *functional attribute type* and *functional attribute value specified by the mobile agent and* returns varying number of components based on the policy associated with the HH. Functional attributes that were used in the experiments are Algorithms, Complexity, and Technology. Functional attribute values that were used for different fuctional attributes are as follows:

- Algorithms: JFC
- Complexity: $O(1)$

- Technology: Java RMI

	PB	PI	RB	RI
State1	8	6	4	4
State2 (Algorithm)	6	4	3	3
State2 (Complexity)	5	3	3	2
State2 (Technology)	7	5	4	3

Table 5.1 Policy Association between HH1 and AR1

	PB	PI	RB	RI
State1	8	6	4	4
State2 (Algorithm)	4	2	2	2
State2 (Complexity)	7	4	3	4
State2 (Technology)	5	3	3	2

Table 5.2 Policy Association between HH1 and AR2

Experiments were conducted to test whether different ARs would give varying results to mobile agents acting on behalf of Headhunters when the ARs are in different states. Table 5.1 and Table 5.2 provide the details of the policy relationship of HH1 with AR1 and AR2 respectively and number of components returned by respective ARs.

Hence, this experiment proved that mobile agents would be able to retrieve component information effectively from the ARs with varying policies.

The results obtained for all the experiments that involve CQRRT measurement, ACM measurement and the heterogeneous policies were as expected. The observations obtained from the above experiments are summarized below:

- An increase in the number of Components will increase the CQRRT. This observation is supported by what is observed in Figure 5.1. As the number of components available in the Meta-repositories increases, all the Headhunters servicing a particular query takes more time to retrieve components from their respective Meta-repositories. Hence, there will be an increase in the CQRRT.

- An increase in the number of Active Registries has negligible effect on the CQRRT. This observation is supported by what is observed in Figure 5.2. The reason is that the component discovery process and the component selection process are independent of each other.

- An increase in the number of Headhunters will result a proportionate increase of the CQRRT. This observation is supported by what is observed in Figure 5.3. The reason is that the mobile agent visits one Headhunter after another and each Headhunter consumes considerable amount of time to process the query. This leads to an increase in the CQRRT.

- An increase in the number of Queries will result a proportionate increase of the CQRRT. This observation is supported by what is observed in Figure 5.4. As the number of incoming queries increases, Headhunters become busy in servicing those requests in addition to the requests that they are still servicing and an increase in the CQRRT will be the result.

- The message consumption of the mobile agent-based component discovery process requires less number of network resources when compared to that of the request-reply based component discovery process.

- Mobile agents would be able to discover components from various Active Registries by providing appropriate credentials to respective Active Registries during the component discovery phase.

This chapter presented the experimentation details to validate the proposed MURDS architecture. The next chapter provides the conclusion of this report and future work.

6. CONCLUSION AND FUTURE WORK

This project presented the architecture and an implementation for a resource discovery service, the "UniFrame Mobile Agent based Resource Discovery Service (MURDS)". The MURDS architecture proposed in this project is based on the URDS architecture [9]. The MURDS prototype was implemented in Java. The MURDS itself forms a part of a framework, "UniFrame", which aims at providing a platform for building DCS by integrating existing and emerging distributed component models under a common meta-model that enables discovery, interoperability, and collaboration of components via generative software techniques. The MURDS prototype was tested on limited machines to validate its functionality and scalability.

The contributions of this project are as follows:

- It provides a survey of the issues associated with the component discovery and component selection phases of the URDS and establishes the need for introducing mobile agents to address these issues in the MURDS architecture.
- It presents a framework for MURDS by adding mobile agents into the URDS architecture.
- It provides an access control that allows mobile agents to get differentiated access to the resources associated with the NRs/LSs in the component discovery phase of the MURDS.
- It compares the perfomance of mobile agent based resource discovery service with the non-mobile agent based resource discovery service to prove that the mobile agent based resource discovery service provides better service when compared to the non-mobile agent based resource discovery service.

Future work to complete for the MURDS involves enhancing the implementation of the prototype and adding more components to the prototype for further testing.

Some future work for the MURDS and the prototype includes:

- The performance of the prototype can be optimized by fine tuning various parameters such as the number of Headhunters present in the system per domain, the time period between periodic component discovery process by the mobile agents, the time period between successive purge cycles to maintain the 'freshness' of the information returned by the mobile agents during the component discovery process, the time period between successive purge cycles to maintain the 'freshness' of the entities participating in the component discovery process and component selection process, etc.

- The mobile agent based communication pattern used for the component discovery and the component selection processes of the MURDS would increases the turn around time as the number of entities involved in the search space increases. Alternative techniques such as dividing the Headhunters/the ActiveRegistries into multiple groups and assigning each group to a mobile agent would reduce the turn around time for these two processes.

- The prototype implementation of the MURDS used a cost based example to show that mobile agents provide an optimal way of discovering components from multiple ActiveRegistries that offer differentiated services to different Headhunters. This idea can be extended to develop a cost-based framework for the discovery aspect of the MURDS.

- Due to the limited number of Windows-OS systems available for experimentation, fewer Headhunters, Active Registries, Query Managers and clients were used than would be optimal for these experiments. Future work would include the use of more systems to run a large number of Headhunters, Active Registries, Query Managers and clients in order to evaluate the scalability of the MURDS prototype.

In conclusion, this project has enhanced the URDS architecture by introducing mobile agents for component discovery and component selection phases. Mobile agents reduced the message consumption for the discovery process by processing the requests at the location of the Active Registries. By making asynchronous calls, mobile agents relieved respective entities, which participate in the component discovery process and the component selection process, from maintaining network connections over an extended

period of time. Mobile agents provided an effective way of discovering components from various Active Registries that offer differentiated services to various Headhunters. The MURDS architecture, coupled with the UniFrame Approach, presents a promising solution for discovery and selection of geographically scattered components.

LIST OF REFERENCES

[1] Orfali R, and Harkey, D. Client/Server Programming with JAVA and CORBA. The second edition. John Wiley & Sons, Inc., 1998.

[2] Microsoft Corporation. DCOM Specifications. http://www.microsoft.com/oledev/olecom, 1998.

[3] Object Management Group. CORBA Components. Technical report, Object Management Group TC Document orbos/99-02-05, March 1999.

[4] .NET, Microsoft Corporation. http://www.microsoft.com/net/. 2003

[5] Quality Vocabulary, ISO, Report: ISO 8402, pp.8.

[6] Raje, R., Auguston, M., Bryant, B. R., Olson, A., Burt, C. "A Unified Approach for the Integration of Distributed Heterogeneous Software Components". Proceedings of the 2001 Monterey Workshop on Engineering Automation for Software Intensive System Integration, Monterey, California, 2001, pp: 109-119.

[7] Raje, R., Auguston, M., Bryant, B. R., Olson, A., Burt, C. "A Quality-of-Service-based Framework for Creating Distributed Heterogeneous Software Components". Concurrency and Computation: Practice and Experience, Volume 14, Issue 12, 2002. Pages: 1009-1034.

[8] Raje, R. "UMM: Unified Meta-object Model for Open Distributed Systems". Proceedings of 4th IEEE International Conference on Algorithms and Architecture for Parallel Processing, ICAP3PP'2000, pp: 454-465, Hong Kong, 2000.

[9] Nanditha N. Siram. "An Architecture for the Uniframe Resource Discovery Service". M. S. Thesis. Department of Computer & Information Science. Indiana University Purdue University Indianapolis, March 2002.

[10] "UDDI Technical White Paper", September 2000.http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.pdf

[11] Object Management Group. "Trading Object Service Specification," Object Management Group 2000. ftp://ftp.omg.org/pub/docs/formal/00-06-27.pdf.

[12] Open LDAP. http://www.openldap.org/jldap/overview.html

[13] ITU/ISO Recommendation X.500 (08/97): Open Systems Interconnection – The Directory: Overview of concepts, models and services. International Telecommunication Union, 1997.

[14] Mockapetris, P. "Domain Names-Implementation and Specification," IETF RFC 1035, October 1987. http://www.rfc-editor.org/rfc/rfc1035.txt

[15] Neeran M. Karnik and Anand R. Tripathi. "Design issues in mobile-agent programming systems". IEEE Concurrency, 6(3): 52--61, 1998.

[16] Danny B. Lange and Mitsuru Oshima. "Seven good reasons for mobile agents". Communications of the ACM, 42(3): 88--89, March 1999.

[17] George Coulouris And Jean Dollimore. "Security Requirements for Cooperative Work: A Model and Its System Implications". Position paper for Sixth SIGOPS European Workshop, Dagstuhl, September 1994.

[18] E.Bierman, E.Cloete. "Classification of malicious host threats in mobile agent computing". Proceedings of the 2002 annual research conference of the South African institute of computer scientists and information technologists on enablement through technology.

[19] Burt, Carol C., Bryant, Barrett R., Raje, Rajeev R., Olson, Andrew M., and Auguston, Mikhail, "Model Driven Security: Unification of Authorization Models for Fine-Grain Access Control", Proceedings of EDOC 2003, The 7th IEEE International Enterprise Distributed Object Computing Conference, September 16 - 19, 2003, Brisbane, Australia.

[20] Zhisheng Huang. "The UniFrame System-Level Generative Programming Framework." M. S. Thesis, Department of Computer & Information Science, Indiana University Purdue University Indianapolis, May, 2003.

[21] G. Brahnmath. "The UniFrame Quality Of Service Framework." M. S. Thesis, Department of Computer & Information Science, Indiana University Purdue University Indianapolis, December, 2002.

[22] G. Brahnmath, R. Raje, A. Olson, B. Bryant, M. Auguston, C.Burt, "A Quality of Service Catalog for Software Components", pp: 513-520, The Proceedings of the Southeastern Software Engineering Conference, Huntsville, Alabama, April 2002.

[23] G. Brahnmath, R. Raje, A. Olson, C. Sun. "Quality of Service Catalog for Software Components ". Technical report (TR-CIS-0219-01), Department of Computer and Information Science, Indiana University Purdue University Indianapolis, 2001.

[24] Coulouris, G., Dollimore, J., Kindberg, T., "Distributed Systems Concepts and Design", Third Edition, Addison-Wesley, 2001.

[25] Sun Microsystems, Jini Specifications V2.0.

[26] Guttman, Erik, "Service Location Protocol: Automatic Discovery of IP Network Services," IEEE Internet Computing, vol. 3, no. 4, pp. 71-80, 1999.

[27] Perkins, C., Guttman, E., "DHCP Options for Service Location Protocol," IETF RFC 2610, June 1999.

[28] Czerwinski, S. E., Zhao, B. Y., Hodes, T. D., Joseph, A. D., Katz, R. H., "An Architecture for a Secure Service Discovery Service," Proceedings of Mobicom '99, 1999.

[29] Ninja, "The Ninja Project," http://ninja.cs.berkeley.edu, 2002.

[30] Salutation Consortium, "Salutation Architecture Specification Version 2.0c –Part 1," The Salutation Consortium, June 1, 1999. http://www.salutation.org

[31] Salutation Consortium, "Salutation Architecture Specification Version 2.0c –Part 2," The Salutation Consortium, June 1, 1999. http://www.salutation.org

[31] Rekesh John, "UPnP, Jini and Salutation - A look at some popular coordination frameworks for future networked devices," California Software Labs, June 17, 1999.

[33] Microsoft Corporation, "Universal Plug and Play Device Architecture Version 1.0," June 8, 2000, www.upnp.org/download/UPnPDA10_20000613.htm

[34] Goland, Y., Cai, T., Leach P., Gu, Y., and Albright, S., "Simple Service Discovery Protocol," IETF, Draft draft-cai-ssdp-v1-03, October 28 1999, http://www.ietf.org/internet-drafts/draft-cai-ssdp-v1-03.txt.

[35] Bluetooth White Paper. http://www.bluetooth.com/developer/whitepaper.

[36] Dipanjan Chakraborty, Filip Perich, Sasikanth Avancha, and Anupam Joshi, "DReggie: A Smart Service Discovery Technique for E-Commerce Applications", In 20th Symposium on Reliable Distributed Systems, October 2001.

[37] DARPA Agent Markup Language. http://www.daml.org/ .

[38] S. Green, L. Hurst, B. Nangle, P. Cunningham, F.Somers and R. Evans. "Software Agents: A Review". Technical report. Trinity College, Dublin, Ireland, May 1997.

[39] White J., "Telescript technology: The foundation of the electronic market place". General Magic white paper 1995.

[40] Grasshopper. http://www.grasshopper.de./

[41] http://agent.cs.dartmouth.edu/general/agenttcl.html

[42] http://www.tryllian.com/development/

[43] http://www.trl.ibm.com/aglets/

[44] http://www.recursionsw.com/products/whitepapers/whitepapers.asp

[45] http://www.cs.umn.edu/Ajanta/

[46] Nguyen T. Giang, Dang T. Tung. "Agent platform evaluation and comparison". June 2002. http://pellucid.ui.sav.sk/TR-2002-06.pdf

[47] http://www.globus.org/ogsa/

[48] Sun Microsystems, "Designing Enterprise Applications with the J2EETM Platform", http://java.sun.com/blueprints/guidelines/designing_enterprise_applications/.

APPENDICES

<u>APPENDIX A: Class Diagrams for the Entity Objects</u>

Component

+setComponentName()   +getComponentName()
+setSubcase)   +getSubcase)
+setDomainName()   +getDomainName()
+setSystemName()   +getSystemName()
+setDescription()   +getDescription()
+setID()   +getID()
+setVersion()   +getVersion()
+setAuthor()   +getAuthor()
+setDate()   +getDate()
+setValidity()   +getValidity()
+setAtomicity()   +getAtomicity()
+setRegistration()   +getRegistration()
+setModel()   +getModel()
+setPurpose()   +getPurpose()
+setAlgorithms()   +getAlgorithms()
+setComplexity()   +getComplexity()
+setRequiredInterfaces()   +getRequiredInterfaces()
+setProvidedInterfaces()   +getProvidedInterfaces()
+setTechnologies()   +getTechnologies()
+setExpectedResources()   +getExpectedResources()
+setDesignPatterns()   +getDesignPatterns()
+setKnownUsages()   +getKnownUsages()
+setAliases()   +getAliases()
+setPreProcessingCollaborators()   +getPreProcessingCollaborators()
+setPostProcessingCollaborators() +getPostProcessingCollaborators()
+setMobility()   +getMobility()
+setSecurity()   +getSecurity()
+setFaultTolerance()   +getFaultTolerance()
+setQoSMetrics()   +getQoSMetrics()
+setQoSLevel()   +getQoSLevel()
+setCost()   +getCost()
+setQualityLevel()   +getQualityLevel()
+setComponentQoS()   +getComponentQoS()
                             +toString()

```
QueryBean

+QueryBean()                    +getRegistrationQuery()
+getComponentNameQuery()        +getModelQuery()
+getSystemNameQuery()           +getComplexityQuery()
+getSubcaseQuery()              +getSecurityQuery()
+getIDQuery()                   +getFaultToleranceQuery()
+getVersionQuery()              +getQoSLevelQuery()
+getAuthorQuery()               +getCostQuery()
+getDateQuery()                 +getQualityLevelQuery()
+getValidityQuery()             +getDomainQuery()
+getAtomicityQuery()            +getMobilityQuery()
+setNumMetrics()                +setNumOffers()
+getHopCount()                  +tokenizeString()
+setHopCount()                  +getQuery()
+getRequestID()                 +setRequestID()
```

APPENDIX B: Class Diagrams for the Data Access Objects
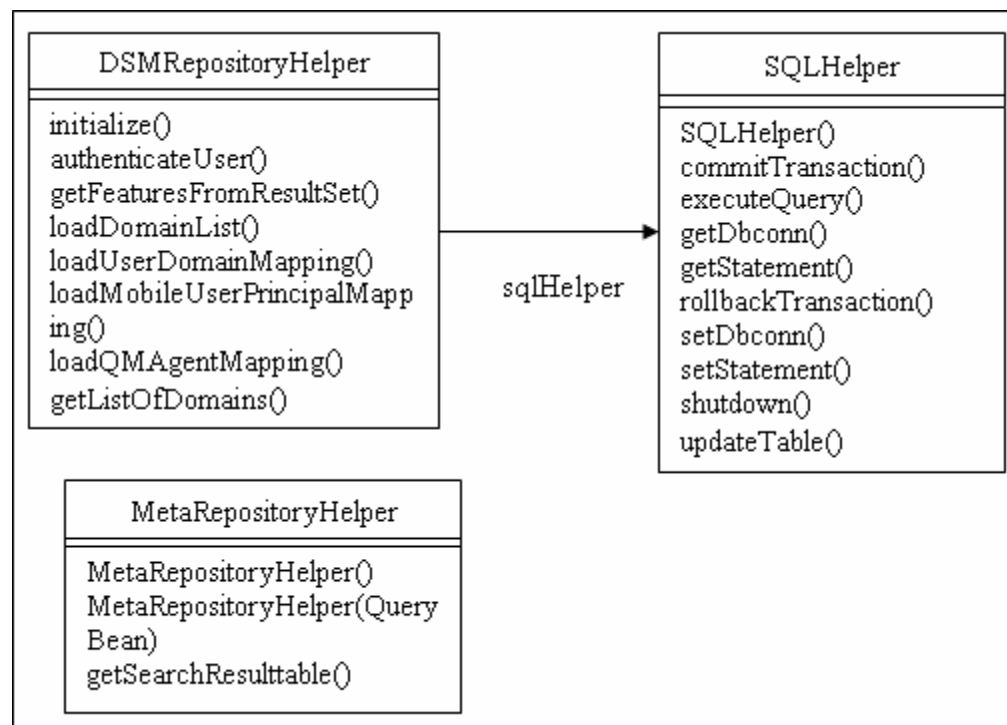
APPENDIX C: Class Diagrams for the Dependent Objects
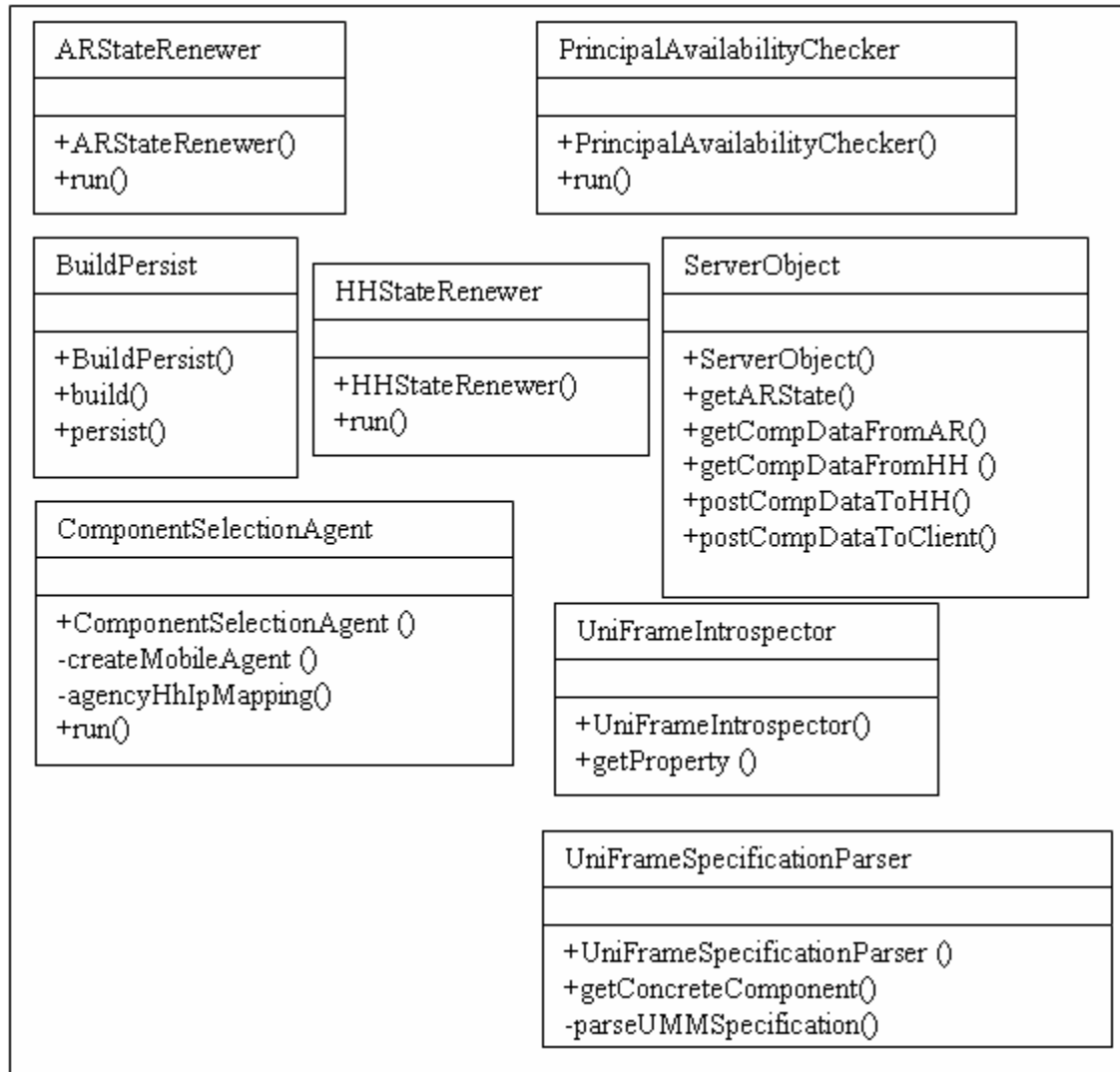
APPENDIX D: Schema for the DSM_Repository

USERS

Column Name	Column Type
USERID	NUMBER
USERTYPE	VARCHAR
USERNAME	VARCHAR
PASSWORD	VARCHAR

USER_PERMISSION_XREF

Column Name	Column Type	
USERID	NUMBER	Foreign Key References USERS(USERID)
PERMISSIONID	NUMBER	Foreign Key References PERMISSIONS(PERMISSIONID)

MOBILE_USERS

Column Name	Column Type
MOBILEUSERID	NUMBER
MOBILEUSERNAME	VARCHAR
MOBILEUSERPASSWORD	VARCHAR
PRINCIPALTYPE	VARCHAR
PRINCIPALNAME	VARCHAR

PERMISSIONS

Column Name	Column Type
PERMISSIONID	NUMBER
PERMISSIONNAME	VARCHAR

APPENDIX E: Schema for the Meta_Repository

UMMSPECIFICATION

Column Name	Column type
COMPONENTNAME	VARCHAR
SUBCASE	VARCHAR
DOMAINNAME	VARCHAR
SYSTEMNAME	VARCHAR
DESCRIPTION	VARCHAR
ID	VARCHAR
VERSION	VARCHAR
AUTHOR	VARCHAR
CREATINGDATE	VARCHAR
VALIDITY	VARCHAR
ATOMICITY	VARCHAR
REGISTRATION	VARCHAR
MODEL	VARCHAR
PURPOSE	VARCHAR
COMPLEXITY	VARCHAR
MOBILITY	VARCHAR
SECURITY	VARCHAR
FAULTTOLERANCE	VARCHAR
QOSLEVEL	VARCHAR
COST	VARCHAR
QUALITYLEVEL	VARCHAR

COMPFUNCQOS

Column Type	Column Name
ID	VARCHAR
COMPONENTNAME	VARCHAR
SYSTEMNAME	VARCHAR
FUNCTIONNAME	VARCHAR
QOSPARAMETER	VARCHAR
VALUE	VARCHAR

EXPECTEDRESOURCES

Column Type	Column Name
ID	VARCHAR
COMPONENTNAME	VARCHAR
DOMAINNAME	VARCHAR
SYSTEMNAME	VARCHAR
EXPECTEDRESOURCE	VARCHAR

APPENDIX F: Class Diagrams for the Service Components
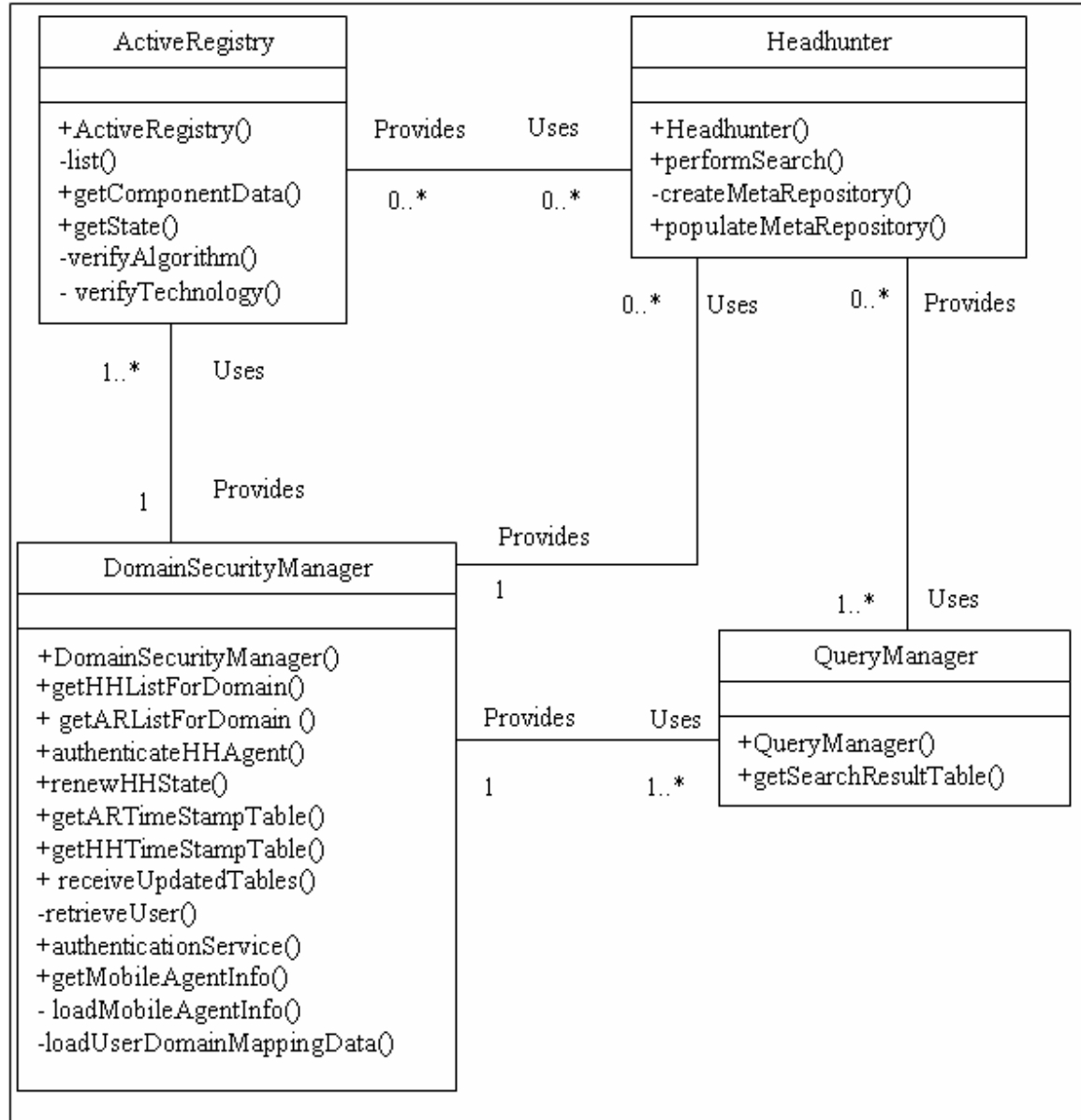
APPENDIX G: Source Code

AbstractComponent.java

```java
public class AbstractComponent extends Component
{
}
```

ActiveRegistry.java

```java
import java.rmi.registry.*;
import java.rmi.*;
import java.net.*;
import java.util.*;
import java.rmi.server.*;
import java.lang.Integer;
import java.lang.String;

/**
 * The active registry class uses the ARStateRenewer to periodically
 * update it's state with the DSM.
 * Creation date: (05/15/2003 10:10:30 AM)
 * @author: Jayasree Gandhamaneni
 */

public class ActiveRegistry extends UnicastRemoteObject implements IActiveRegistry
{

        private int port = 9000;
        private String userType = "Registry";
        private String rmiLocn =null;
        private IDomainSecurityManager dsmanager = null;

        /**
         * Obtain URL List of Registered components.
         */

        private String[] list(int argPort,String rmiLocn)
        {
                String[] listOfURLS = null;
                try
                {
                        listOfURLS = Naming.list("//"+rmiLocn+":" + argPort);
                } catch (java.rmi.RemoteException e) {
                        System.out.println(e.getMessage());
                } catch (Exception e) {
                        System.out.println(e.getMessage());
                }

                return listOfURLS;
        }

        public static void main(String[] args) {
                long renewalTime = 30000;
```

```
            int rmiRegistryPort = 0000;
            String activeRegistryLocation="//"+args[0]+":"+args[1] + "/ActiveRegistry";
            rmiRegistryPort = Integer.parseInt(args[2].trim());
            System.out.println("Port in use is"+rmiRegistryPort);
            String dsmLocation ="//"+args[3]+":"+ args[4]+"/DomainSecurityManager";
            String domain = args[5];
            String userName = args[6];
            String password = args[7];

            try {
                    System.setSecurityManager(new RMISecurityManager());
                    Naming.rebind(
                            activeRegistryLocation,
                            new ActiveRegistry(
                                    renewalTime,
                                    rmiRegistryPort,
                                    userName,
                                    password,
                                    domain,
                                    activeRegistryLocation,
                                    dsmLocation)
                            );
                    System.out.println("ActiveRegistry is ready.");

            } catch (Exception e) {
                    System.out.println("ActiveRegistry failed: " + e);
            }
    }

    /**
    * The ActiveRegistry Constructor.
    */
    public ActiveRegistry(
            long rTime,
            int rmiRegistryPort,
            String userName,
            String password,
            String domain,
            String activeRegistryLocation,
            String dsmLocation)
            throws RemoteException {
            try {
                    port = rmiRegistryPort;
int j=0;

                    for (int i=0;i<activeRegistryLocation.length();i++)
                    {
            if(activeRegistryLocation.charAt(i)==':')
            {
                    j =i;
                    i = activeRegistryLocation.length();
            }
                    }

                    rmiLocn=activeRegistryLocation.substring(2,j);
                    LocateRegistry.createRegistry(port);
```

```
                        System.out.println("\n Active Registry Created RMI Registry At : " + port);

                        dsmanager =        (IDomainSecurityManager) Naming.lookup(dsmLocation);

                        System.out.println("Active Registry Contacting DSM for Authentication.");

                        if(dsmanager.authenticationService(
                                userType,
                                userName,
                                password,
                                activeRegistryLocation,
                                domain)) {

                                System.out.println("Active Registry Authenticated by DSM.");

                                ARStateRenewer arStateRenewer = new
ARStateRenewer(rTime,dsmLocation,activeRegistryLocation,domain);
                                Thread renewerThread = new Thread(arStateRenewer);
                                renewerThread.start();
                        }
                        else
                        {
                                System.out.println("Active registry is not a valid principal.
Authentication failed");
                                System.exit(0);
                        }

                } catch (Exception e) {
                        System.out.println("Exception in the constructor of Active
Registry"+e.getMessage());
                }

        }

        /**
        * Returns ActiveRegistry state to HHAgent
        */
        public int getState() throws RemoteException
        {
                Random r = new Random();
                return (r.nextInt(2)+1);
        }

        /**
        * Returns components to HHAgent
        */
        public Hashtable getComponentData(String headhunterLocation, String mobileAgentUserName,
String attributeType, String attributeValue) throws RemoteException
        {
                String serviceType = dsmanager.authenticateHHAgent(headhunterLocation,
mobileAgentUserName);
                Hashtable objectTable = new Hashtable();
                Hashtable level1ObjectTable = new Hashtable();
                Hashtable level2ObjectTable = new Hashtable();
```

```
                System.out.println("AR contacted by "+headhunterLocation+" to Retrieve Component
Data");

            if(serviceType!="") {

                    System.out.println("Headhunter requests for "+serviceType+" services from
AR");

                    try
                {
                        String[] objURL = list(port,rmiLocn);

                        String accessLevel=new String();
                        if(serviceType.equals("PB"))
                                accessLevel= "ALL";
                        else if(serviceType.equals("PI"))
                                accessLevel= "L1L2";
                        else if(serviceType.equals("RB"))
                                accessLevel= "L1";
                        else if(serviceType.equals("RI"))
                                accessLevel= "L2";
                        System.out.println("Access level is "+accessLevel);
                        System.out.println("Attribute type -"+attributeType+" Attribute Value -
"+attributeValue);

                        for(int i=0;i<objURL.length;i++)
                        {
                                System.out.println("Active Registry gathering component
information from : " + objURL[i]);

                                //Registry looking up object registered with it.
                                System.setSecurityManager(new RMISecurityManager());
                                Object  obj = Naming.lookup(objURL[i]);

                                //Obtain the location(URL) of the UMMSpecification for this
object by

                                //introspecting its ummSpecification property.

                                String ummSpecURL = (String)
UniFrameIntrospector.getProperty(obj,"ummSpecURL");

                                //Call the XMLParser by passing this URL. The XML Parser
will parse the XML specification

                                //and construct a Component from the specification which it
returns.

                                UniFrameSpecificationParser xmlDomParser = new
UniFrameSpecificationParser(ummSpecURL);
                                ConcreteComponent component =
xmlDomParser.getConcreteComponent();

                                if(attributeType.equals("") && attributeValue.equals(""))
                                {
                                        if(accessLevel.equals("ALL"))
                                                objectTable.put(objURL[i],component);
                                        else if(accessLevel.equals("L1L2"))
```

```
                                                                    {
                                                                            if((component.getCost()).equals("L1"))

        level1ObjectTable.put(objURL[i],component);
                                                                            else if((component.getCost()).equals("L2"))

        level2ObjectTable.put(objURL[i],component);
                                                                    }
                                                                    else if((component.getCost()).equals(accessLevel))
                                                                    {

                                                                            objectTable.put(objURL[i],component);
                                                                    }

                                                            }
                                                    else if(attributeType.equals("algorithm"))
                                                    {
                                                            String algorithms []=component.getAlgorithms();
                                                            if(accessLevel.equals("ALL") &&
verifyAlgorithm(algorithms,attributeValue))

                                                            {
                                                                     objectTable.put(objURL[i],component);

                                                            }
                                                            else if(accessLevel.equals("L1L2") &&
verifyAlgorithm(algorithms,attributeValue))

                                                            {
                                                                    if((component.getCost()).equals("L1"))

        level1ObjectTable.put(objURL[i],component);
                                                                    else if((component.getCost()).equals("L2"))

        level2ObjectTable.put(objURL[i],component);

                                                            }
                                                            else
if((component.getCost()).equals(accessLevel)&& verifyAlgorithm(algorithms,attributeValue))
                                                                    {
                                                                            objectTable.put(objURL[i],component);
                                                                    }
                                                            }
                                                    else if(attributeType.equals("complexity"))
                                                    {
                                                            if(accessLevel.equals("ALL") &&
(component.getComplexity()).equals(attributeValue))
                                                            {

                                                                    objectTable.put(objURL[i],component);
                                                            }
                                                            else if(accessLevel.equals("L1L2") &&
(component.getComplexity()).equals(attributeValue))
                                                            {
                                                                    if((component.getCost()).equals("L1"))

        level1ObjectTable.put(objURL[i],component);
                                                                            else if((component.getCost()).equals("L2"))
```

```
        level2ObjectTable.put(objURL[i],component);

                                        }
                                        else if((component.getCost()).equals(accessLevel)
&& (component.getComplexity()).equals(attributeValue))
                                        {

                                                objectTable.put(objURL[i],component);
                                        }
                                }
                                else if(attributeType.equals("technology"))
                                {
                                        String technologies []=component.getTechnologies();
                                        if(accessLevel.equals("ALL") &&
verifyTechnology(technologies,attributeValue))
                                        {
                                                objectTable.put(objURL[i],component);
                                        }
                                        else if(accessLevel.equals("L1L2") &&
verifyTechnology(technologies,attributeValue))
                                        {
                                                if((component.getCost()).equals("L1"))

        level1ObjectTable.put(objURL[i],component);
                                                else if((component.getCost()).equals("L2"))

        level2ObjectTable.put(objURL[i],component);
                                        }
                                        else if((component.getCost()).equals(accessLevel)
&& verifyTechnology(technologies,attributeValue))
                                        {
                                                objectTable.put(objURL[i],component);
                                        }
                                }
                        }//end for

                        if(accessLevel.equals("L1L2"))
                        {
                                if(level1ObjectTable.size()>0)
                                {
                                        int i=0;
                                        Enumeration e=level1ObjectTable.keys();
                                        while(i <= (level1ObjectTable.size()/2)){
                                                String urlID=(String)e.nextElement();

        objectTable.put(urlID,(ConcreteComponent)level1ObjectTable.get(urlID));
                                                i++;

                                        }

                                }

                                if(level2ObjectTable.size()>0)
                                {
                                        int j=0;
```

```
                                              Enumeration e=level2ObjectTable.keys();
                                              while(j <= (level2ObjectTable.size()/2)){
                                                    String urlID=(String)e.nextElement();

        objectTable.put(urlID,(ConcreteComponent)level2ObjectTable.get(urlID));
                                                    j++;
                                              }
                                        }
                                  }
                        } catch(Exception e){
                                    System.out.println(e.getMessage());
                        }
                  }
                  else {
                                    System.out.println("Mobile agent is not a valid member to access
component data");
                                    System.out.println("Access to ActiveRegistry rejected");

                  }
                  return objectTable;
        }

        /**
        * Checks the algorithm type of a component.
        */
        private boolean verifyAlgorithm(String[] algorithms, String attributeValue)
        {
                  boolean isVerified = false;
                  if(algorithms!=null)
                  {
                          int i=0;
                          for(i=0;i<algorithms.length;i++)
                          {
                                    if(algorithms[i].equals(attributeValue))
                                    {
                                              isVerified = true;
                                              i=algorithms.length;
                                    }
                          }
                  }
                  return isVerified;

        }

        /**
        * Checks the technology used for the development of a component.
        */
        private boolean verifyTechnology(String[] technologies, String attributeValue)
        {
                  boolean isVerified = false;
                  if(technologies!=null)
                  {
                          int i=0;
                          for(i=0;i<technologies.length;i++)
                          {
                                    if(technologies[i].equals(attributeValue))
```

```
                                    {
                                            isVerified = true;
                                            i=technologies.length;
                                    }
                            }
                    }
                    return isVerified;
            }


}//end of Active Registry
```

ARStateRenewer.java

```java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;
import java.util.*;

/**
 * This class operates as a Thread which executes periodically
 * to update availability of an active registry with the
 * DomainSecurityManager.
 * Creation date: (06/15/2003 10:10:30 AM)
 * @author: Jayasree Gandhamaneni
 */

public class ARStateRenewer implements Runnable
{
        private long rTime = 0;
        private String arLocation = "";
        private IDomainSecurityManager dsm = null;
        private String arDomain = "";


        /**
        * The ARStateRenewer Constructor.
        */
        public ARStateRenewer(
                long renewalTime,
                String dsmLocation,
                String activeRegistryLocation,
                String domain ) {

                try {

                        System.out.println("ARStateRenewer thread starting active registry state
updation with the DSM");
                        System.setSecurityManager(new RMISecurityManager());
                        dsm = (IDomainSecurityManager) Naming.lookup(dsmLocation);
                        rTime = renewalTime;
                        arLocation = activeRegistryLocation;
                        arDomain = domain;

                }catch (Exception e){
```

```
                    System.out.println("Exception in the constructor of ActiveRegistryStateRenewer
"+e);
                }
        }

        public void run()
        {
                Thread CurrentThread = Thread.currentThread();

                try {
                        while(true)
                        {
                                dsm.renewARState(arLocation, arDomain);
                                //System.out.println("AR state updated with DSM");
                                CurrentThread.sleep(rTime);
                        }
                }catch(InterruptedException ie) {
                        System.out.println(ie.getMessage());
                }
                catch (Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

buildpersist.java

```
import java.util.*;
import java.sql.*;
import java.io.*;
import java.lang.*;

public class buildpersist implements Serializable{

 public buildpersist(){}

/**
* Retrieves component data from the Meta_Repository.
*/
 public ConcreteComponent build(ConcreteComponent component, ResultSet result, String
componentTablename)
 {
        try{
                component.setComponentName(result.getString("componentName"));
                component.setSubcase(result.getString("subcase"));
                component.setDomainName(result.getString("domainName"));
                component.setSystemName(result.getString("systemName"));
                component.setDescription(result.getString("description"));
                component.setID(result.getString("id"));
                component.setVersion(result.getString("version"));
                component.setAuthor(result.getString("Author"));
                component.setDate(result.getString("CreatingDate"));
                component.setValidity(result.getString("validity"));
                component.setAtomicity(result.getString("atomicity"));
                component.setRegistration(result.getString("registration"));
```

```java
                component.setModel(result.getString("Model"));
                component.setPurpose(result.getString("Purpose"));
                component.setComplexity(result.getString("complexity"));
                component.setMobility(result.getString("mobility"));
                component.setSecurity(result.getString("security"));
                component.setFaultTolerance(result.getString("faultTolerance"));
                component.setQoSLevel(result.getString("qosLevel"));
                component.setCost(result.getString("cost"));
                component.setQualityLevel(result.getString("qualityLevel"));
        }catch (Exception e) {
                System.out.println("Exception in building component" + e.getMessage());
        }
        try {
                SQLHelper sqlEngine = new SQLHelper();

                try {
                        String algorithmsQuery = "SELECT * from "+ componentTablename +
                                                        "Algorithms where id =
'"+component.getID() +
                                                        "' AND " + " ComponentName ='"
+component.getComponentName() +
                                                        "' AND " + "DomainName = '" +
component.getDomainName() +
                                                        "' AND " + "SystemName = '" +
component.getSystemName() + "'";
        ResultSet algorithmsResult = sqlEngine.executeQuery(algorithmsQuery);
        ArrayList algorithms = new ArrayList();
        while(algorithmsResult.next())
        {
                algorithms.add(algorithmsResult.getString("Algorithm"));
        }
        if(algorithms.size() != 0)
        {
                component.setAlgorithms((String[])algorithms.toArray(new String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in algorithms query retrieval"+e.getMessage());
                }

                try {
                        String requiredInterfacesQuery = "SELECT * from  "+ componentTablename +
                                                        "RequiredInterfaces where Id =
'"+component.getID() +
                                                        "' AND " + " ComponentName =
'" + component.getComponentName() +
                                                        "' AND " + "DomainName = '" +
component.getDomainName() +
                                                        "' AND " + "SystemName = '" +
component.getSystemName() + "'";
        ResultSet requiredInterfacesResult = sqlEngine.executeQuery(requiredInterfacesQuery);
        ArrayList requiredInterfaces = new ArrayList();
        while(requiredInterfacesResult.next())
        {
                        requiredInterfaces.add(requiredInterfacesResult.getString("Interface"));
        }
        if(requiredInterfaces.size() != 0)
```

```
        {
                component.setRequiredInterfaces((String[])requiredInterfaces.toArray(new String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in required interfaces query retrieval"+e.getMessage());
                }

                try {
                        String providedInterfacesQuery = "SELECT * from  "+ componentTablename +
                                                        "ProvidedInterfaces where Id =
'"+component.getID()+
                                                        "' AND " + " ComponentName =
'" + component.getComponentName() +
                                                        "' AND " + "DomainName = '" +
component.getDomainName() +
                                                        "' AND " + "SystemName = '" +
component.getSystemName() + "'";
        ResultSet providedInterfacesResult = sqlEngine.executeQuery(providedInterfacesQuery);
        ArrayList providedInterfaces = new ArrayList();
        while(providedInterfacesResult.next())
        {
                providedInterfaces.add(providedInterfacesResult.getString("Interface"));
        }
        if(providedInterfaces.size() != 0)
        {
                component.setProvidedInterfaces((String[])providedInterfaces.toArray(new String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in provided interfaces query
retrieval"+e.getMessage());
                }

                try {
                        String technologiesQuery = "SELECT * from  "+ componentTablename +
                                                        "Technologies where Id =
'"+component.getID()+
                                                        "' AND " + " ComponentName = '" +
component.getComponentName() +
                                                        "' AND " + "DomainName = '" +
component.getDomainName() +
                                                        "' AND " + "SystemName = '" +
component.getSystemName() + "'";
        ResultSet technologiesResult = sqlEngine.executeQuery(technologiesQuery);
        ArrayList technologies = new ArrayList();
        while(technologiesResult.next())
                {
                        technologies.add(technologiesResult.getString("Technology"));
        }
        if(technologies.size() != 0)
        {
                        component.setTechnologies((String[])technologies.toArray(new String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in technologies query retrieval"+e.getMessage());
                }
```

```
                try {
                        String expectedResourcesQuery = "SELECT * from "+ componentTablename +
                                                      "ExpectedResources where Id =
'"+component.getID()+
                            "' AND " + " ComponentName = '" + component.getComponentName() +
                                                      "' AND " + "DomainName = '" +
component.getDomainName() +
                                                      "' AND " + "SystemName = '" +
component.getSystemName() + "'";
        ResultSet expectedResourcesResult = sqlEngine.executeQuery(expectedResourcesQuery);
        ArrayList expectedResources = new ArrayList();
        while(expectedResourcesResult.next())
        {
                expectedResources.add(expectedResourcesResult.getString("ExpectedResource"));
        }
        if(expectedResources.size() != 0)
        {
                component.setExpectedResources((String[])expectedResources.toArray(new String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in expected resources retrieval"+e.getMessage());
                }

                try {
                String designPatternsQuery = "SELECT * from "+ componentTablename +
                                                      "DesignPatterns where Id =
'"+component.getID()+
                                                      "' AND " + " ComponentName = '" +
component.getComponentName() +
                                                      "' AND " + "DomainName = '" +
component.getDomainName() +
                                                      "' AND " + "SystemName = '" +
component.getSystemName() + "'";
                ResultSet designPatternsResult = sqlEngine.executeQuery(designPatternsQuery);
        ArrayList designPatterns = new ArrayList();
        while(designPatternsResult.next())
        {
                designPatterns.add(designPatternsResult.getString("Pattern"));
        }
        if(designPatterns.size() != 0)
        {
                component.setDesignPatterns((String[])designPatterns.toArray(new String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in design patterns query retrieval"+e.getMessage());
                }

                try {
                String knownUsagesQuery = "SELECT * from "+ componentTablename +
                                                      "KnownUsages where Id =
'"+component.getID()+
                                                      "' AND " + " ComponentName = '" +
component.getComponentName() +
                                                      "' AND " + "DomainName = '" +
component.getDomainName() +
                                                      "' AND " + "SystemName = '" +
```

```
component.getSystemName() + "'";
        ResultSet knownUsagesResult = sqlEngine.executeQuery(knownUsagesQuery);
        ArrayList knownUsages = new ArrayList();
        while(knownUsagesResult.next())
        {
           knownUsages.add(knownUsagesResult.getString("Usage"));
        }
        if(knownUsages.size() != 0)
                {
                component.setKnownUsages((String[])knownUsages.toArray(new String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in known usages retrieval"+e.getMessage());
                }

                try {
                String aliasesQuery = "SELECT * from  "+ componentTablename +
                                                "Aliases where Id = '"+component.getID()+
                        "' AND " + " ComponentName = '" + component.getComponentName() +
                                                "' AND " + "DomainName = '" +
component.getDomainName() +
                                                "' AND " + "SystemName = '" +
component.getSystemName() + "'";
        ResultSet aliasesResult = sqlEngine.executeQuery(aliasesQuery);
        ArrayList aliases = new ArrayList();
        while(aliasesResult.next())
        {
                aliases.add(aliasesResult.getString("Alias"));
        }
        if(aliases.size() != 0)
        {
                component.setAliases((String[])aliases.toArray(new String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in aliases query retrieval"+e.getMessage());
                }

                try {
                String preProcessingCollaboratorsQuery = "SELECT * from  "+ componentTablename +
                                                "PreProcessing where Id
= '"+component.getID()+
                                                "' AND " + "
ComponentName = '" + component.getComponentName() +
                                                "' AND " +
"DomainName = '" + component.getDomainName() +
                                                "' AND " +
"SystemName = '" + component.getSystemName() + "'";
        ResultSet preProcessingCollaboratorsResult =
sqlEngine.executeQuery(preProcessingCollaboratorsQuery);
        ArrayList preProcessingCollaborators = new ArrayList();
        while(preProcessingCollaboratorsResult.next())
        {

        preProcessingCollaborators.add(preProcessingCollaboratorsResult.getString("Collaborator"));
        }
        if(preProcessingCollaborators.size() != 0)
```

```
        {

        component.setPreProcessingCollaborators((String[])preProcessingCollaborators.toArray(new
String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in preprocessing query retrieval"+e.getMessage());
                }

                try {
                        String postProcessingCollaboratorsQuery = "SELECT * from "+
componentTablename +
                                                                "PostProcessing
where Id = '"+component.getID()+
                                                                "' AND " + "
ComponentName = '" + component.getComponentName() +
                                                                "' AND " +
"DomainName = '" + component.getDomainName() +
                                                                "' AND " +
"SystemName = '" + component.getSystemName() + "'";
        ResultSet postProcessingCollaboratorsResult =
sqlEngine.executeQuery(postProcessingCollaboratorsQuery);
        ArrayList postProcessingCollaborators = new ArrayList();
        while(postProcessingCollaboratorsResult.next())
        {

        postProcessingCollaborators.add(postProcessingCollaboratorsResult.getString("Collaborator"));
        }
        if(postProcessingCollaborators.size() != 0)
        {

        component.setPostProcessingCollaborators((String[])postProcessingCollaborators.toArray(new
String[1]));
        }
                }catch(Exception e){
                        System.out.println("Error in postprocessing query retrieval"+e.getMessage());
                }

                // Building ComponentQoS object
                //Get all the entries from the table
                /*try
                {
                        SQLHelper sqlengine1 = new SQLHelper();
                        SQLHelper sqlengine2 = new SQLHelper();
                        SQLHelper sqlengine3 = new SQLHelper();
                        String componentqosString="SELECT * FROM
"+componentTablename+"CompFuncQoS";
                        ResultSet answerset1, oneuse;
                        // oneuse is used for getting the systemname and componentname the first time
                        oneuse = sqlEngine.executeQuery(componentqosString);
                        oneuse.next();
                        // Scroll through the entries one row at a time
                        String cname = oneuse.getString("componentName");
                        String sname = oneuse.getString("systemName");
                        ComponentQoS qoscomponent = new ComponentQoS(sname, cname);
                        answerset1 = sqlengine1.executeQuery(componentqosString);
```

```
                        while(answerset1.next())
                        {
                                String componentname = "";
                                String systemname    = "";
                                String functionname = "";
                                String qosparameter = "";
                                String value        = "";
                                try {
                                componentname =   answerset1.getString("componentName");
                                systemname    = answerset1.getString("systemName");
                                        functionname  = answerset1.getString("functionname");
                                qosparameter  = answerset1.getString("qosparameter");
                                value         = answerset1.getString("value");
                                }catch(Exception e) {
                                        System.out.println("Error in getting parameters from result set :
build --> buildpersist "+e.getMessage());
                                }

                                // For testing
                                System.out.println(componentname +systemname + functionname +
qosparameter + value);
                                // The systemname is same for every record, but the componentname
changes
                                // So...scroll through the records with the primary index being
componentname
                                qoscomponent = new ComponentQoS(systemname,componentname);
                                ResultSet answerset2 = sqlengine2.executeQuery(componentqosString);
                                while(answerset2.next())
                                {
                                try{
                                        // Get the function name and create object functionqos.
                                        functionname = answerset2.getString("functionname");
                                        System.out.println(" answerset" +functionname);
                                        FunctionQoS qosfunction = new
FunctionQoS(componentname, functionname);
                                        // For each function, get qosparameter and value in the form of
a hashtable
                                        System.out.println("before answerset3" +functionname);
                                        ResultSet answerset3 =
sqlengine3.executeQuery(componentqosString);
                                        // For testing
                                        System.out.println("after answerset3" +functionname);
                                        while(answerset3.next())
                                        {
                                                functionname =
answerset3.getString("functionname");
                                                qosparameter = answerset3.getString("qosparameter");
                                                value = answerset3.getString("value");
                                                qosfunction.addFunctionQoS(componentname,
functionname, qosparameter, value);
                                                // For testing
                                                System.out.println("Qos stuff" +componentname
+functionname +qosparameter +(String)value);
                                        }//while
                                        answerset3.close();
```

```
                                                //Add the functionqos object to qoscomponent's hashtable
                                                qoscomponent.addFunctionQoS(qosfunction);
                                        }catch(Exception e){
                                                        System.out.println("Error in retrieving object
functionqos from table"+e.getMessage());
                                                }
                                        }//while
                                answerset2.close();
                                }//while. Finally got the ComponentQoS object. Now set it in the
concretecomponent.
                                component.setComponentQoS(qoscomponent);

                }catch(Exception e) {
                                System.out.println("Error in retrieving from tables for results of query
"+e.getMessage());
                        }*/
        } catch(Exception e){
                        System.out.println("Error creating SQLHelper :"+e.getMessage());
        }

        return component;
}

/**
 * Stores component data in the Meta_Repository.
 */
public void persist(
        ConcreteComponent component,
        SQLHelper sqlHelper,
        String componentTableName) throws java.lang.Exception {

        ComponentQoS qoscomponent = component.getComponentQoS();
        FunctionQoS qosfunction;

        if (component !=null)
        {
                String insertString_UMMSpecification = "INSERT INTO " + componentTableName
+"UMMSpecification (" +
                "ComponentName, SubCase, DomainName, SystemName, Description, Id, Version, Author,
" +
                "CreatingDate, Validity, Atomicity, Registration, Model, Purpose, " +
                "Complexity, Mobility, Security, FaultTolerance, QoSLevel, Cost, QualityLevel)" +
                "VALUES('" + component.getComponentName() + "' , '" +
                        component.getSubcase() + "' , '" +
                        component.getDomainName() + "' , '" +
                        component.getSystemName() + "' , '" +
                        component.getDescription() + "' , '" +
                        component.getID() + "' , '" +
                        component.getVersion() + "' , '" +
                        component.getAuthor() + "' , '" +
                        component.getDate() + "' , '" +
                        component.getValidity() + "' , '" +
                        component.getAtomicity() + "' , '" +
                        component.getRegistration() + "' , '" +
                        component.getModel() + "' , '" +
                        component.getPurpose() + "' , '" +
```

```
                    component.getComplexity() + "' , '" +
                    component.getMobility() + "' , '" +
                    component.getSecurity() + "' , '" +
                    component.getFaultTolerance() + "' , '" +
                    component.getQoSLevel() + "' , '" +
                    component.getCost() + "' , '" +
                    component.getQualityLevel() + "')";

            sqlHelper.updateTable(insertString_UMMSpecification);
            // For finding out whether tables have been created
            // For testing purposes
            //System.out.println("Testing ...Table UMMSpec ");
            //for (int in=0;in<insertString_UMMSpecification.length();in++)
            //System.out.println(insertString_UMMSpecification);

            String[] algorithms = component.getAlgorithms();
        if(algorithms != null)
        {
         for (int i = 0; i < algorithms.length; i++)
            {
                String insertString_Algorithms = "INSERT INTO " +componentTableName+
"Algorithms (" +
                "Id, ComponentName, DomainName, SystemName, Algorithm)" +
                "VALUES('" + component.getID() +"' , '" +
                    component.getComponentName() + "' , '" +
                    component.getDomainName() + "' , '" +
                    component.getSystemName() + "' , '" +
                    algorithms[i] + "')";
             sqlHelper.updateTable(insertString_Algorithms);
            }
         }


                try
                {
                        String[] requiredInterfaces = component.getRequiredInterfaces();
                if(requiredInterfaces != null)
                {
                for(int i = 0; i < requiredInterfaces.length; i++)
                {
                String insertString_RequiredInterfaces = "INSERT INTO "+componentTableName+
"RequiredInterfaces (" +
                "Id, ComponentName, DomainName, SystemName, Interface)" +
                "VALUES('" + component.getID() +"' , '" +
                    component.getComponentName() + "' , '" +
                    component.getDomainName() + "' , '" +
                    component.getSystemName() + "' , '" +
                    requiredInterfaces[i] + "')";
                sqlHelper.updateTable(insertString_RequiredInterfaces );
                }
                }
                }catch(Exception e) {
                        System.out.println("insertion in table reqd interfaces problem"+e.getMessage());
                }

                try
```

```
                {
                        String[] providedInterfaces = component.getProvidedInterfaces();
                if(providedInterfaces != null)
                {
                for(int i = 0; i < providedInterfaces.length; i++)
                {
                        String insertString_ProvidedInterfaces = "INSERT INTO
"+componentTableName+"ProvidedInterfaces (" +
                "Id, ComponentName, DomainName, SystemName, Interface)" +
                "VALUES('" + component.getID() +"' , '" +
                    component.getComponentName() + "' , '" +
                    component.getDomainName() + "' , '" +
                    component.getSystemName() + "' , '" +
                    providedInterfaces[i] + "')";
                sqlHelper.updateTable(insertString_ProvidedInterfaces);
                }
                }
                }catch(Exception e) {
                        System.out.println("insertion in table provided interfaces
problem"+e.getMessage());
                }

                try
                {
                        String[] technologies = component.getTechnologies();
                if(technologies != null)
                {
                for(int i = 0; i < technologies.length; i++)
                {
                        String insertString_Technologies = "INSERT INTO "
+componentTableName+"Technologies (" +
                "Id, ComponentName, DomainName, SystemName, Technology)" +
                "VALUES('" + component.getID() +"' , '" +
                    component.getComponentName() + "' , '" +
                    component.getDomainName() + "' , '" +
                    component.getSystemName() + "' , '" +
                    technologies[i] + "')";
                sqlHelper.updateTable(insertString_Technologies);
                }
                }
                }catch(Exception e) {
                        System.out.println("insertion in table technologies problem"+e.getMessage());
                }

                try
                {
                        String[] expectedResources = component.getExpectedResources();
                if(expectedResources != null)
                {
                for(int i = 0; i < expectedResources.length; i++)
                {
                String insertString_ExpectedResources = "INSERT INTO "
+componentTableName+"ExpectedResources (" +
                "Id, ComponentName, DomainName, SystemName, ExpectedResource)" +
                "VALUES('"+ component.getID() +"' , '" +
                    component.getComponentName() + "' , '" +
```

```
                                  component.getDomainName() + "' , '" +
                                  component.getSystemName() + "' , '" +
                                  expectedResources[i] + "')";
                      sqlHelper.updateTable(insertString_ExpectedResources);
                      }
                      }
              }catch(Exception e) {
                          System.out.println("insertion in table expected resources
problem"+e.getMessage());
              }

              try {
                          String[] designPatterns = component.getDesignPatterns();
              if(designPatterns != null)
              {
              for(int i = 0; i < designPatterns.length; i++)
              {
              String insertString_DesignPatterns = "INSERT INTO
"+componentTableName+"DesignPatterns (" +
              "Id, ComponentName, DomainName, SystemName, Pattern)" +
              "VALUES('" + component.getID() +"' , '" +
                  component.getComponentName() + "' , '" +
                  component.getDomainName() + "' , '" +
                  component.getSystemName() + "' , '" +
                  designPatterns[i] + "')";
              sqlHelper.updateTable(insertString_DesignPatterns);
              }
              }
              }catch(Exception e) {
                          System.out.println("insertion in table design patterns problem"+e.getMessage());
              }

              try {
                          String[] knownUsages = component.getKnownUsages();
              if(knownUsages != null)
              {
              for(int i = 0; i < knownUsages.length; i++)
              {
                 String insertString_KnownUsages = "INSERT INTO
"+componentTableName+"KnownUsages (" +
                  "Id, ComponentName, DomainName, SystemName, Usage)" +
                  "VALUES('" + component.getID() +"' , '" +
                  component.getComponentName() + "' , '" +
                  component.getDomainName() + "' , '" +
                  component.getSystemName() + "' , '" +
                  knownUsages[i] + "')";
                 sqlHelper.updateTable(insertString_KnownUsages);
              }
              }
              }catch(Exception e) {
                          System.out.println("insertion in table known usages problem"+e.getMessage());
              }

              try {
                          String[] aliases = component.getAliases();
              if(aliases != null)
```

```
                {
                for(int i = 0; i < aliases.length; i++)
                {
                  String insertString_Aliases = "INSERT INTO "+componentTableName+"Aliases (" +
                    "Id, ComponentName, DomainName, SystemName, Alias)" +
                    "VALUES('" + component.getID() +"' , '" +
                    component.getComponentName() + "' , '" +
                    component.getDomainName() + "' , '" +
                    component.getSystemName() + "' , '" +
                    aliases[i] + "')";
                  sqlHelper.updateTable(insertString_Aliases);

                }
                }
                }catch(Exception e) {
                        System.out.println("insertion in table aliases problem"+e.getMessage());
                }

                try {
                        String[] preProcessingCollaborators =
component.getPreProcessingCollaborators();
                if(preProcessingCollaborators != null)
                {
                 for(int i = 0; i < preProcessingCollaborators.length; i++)
                 {
                   String insertString_PreProcessingCollaborators = "INSERT INTO
"+componentTableName+"PreProcessing (" +
                     "Id, ComponentName, DomainName, SystemName, Collaborator)" +
                     "VALUES('" + component.getID() +"' , '"  +
                   component.getComponentName() + "' , '" +
                   component.getDomainName() + "' , '" +
                   component.getSystemName() + "' , '" +
                   preProcessingCollaborators[i] + "')";
                  sqlHelper.updateTable(insertString_PreProcessingCollaborators);
                }
                }
                }catch(Exception e) {
                        System.out.println("insertion in table preprocessing problem"+e.getMessage());
                }

                try {
                        String[] postProcessingCollaborators =
component.getPostProcessingCollaborators();
                if(postProcessingCollaborators != null)
                {
                        for(int i = 0; i < postProcessingCollaborators.length; i++)
                {
                        String insertString_PostProcessingCollaborators = "INSERT INTO
"+componentTableName+"PostProcessing (" +
                "Id, ComponentName, DomainName, SystemName, Collaborator)" +
                "VALUES('" + component.getID() +"' , '" +
                    component.getComponentName() + "' , '" +
                    component.getDomainName() + "' , '" +
                    component.getSystemName() + "' , '" +
                    postProcessingCollaborators[i] + "')";
                sqlHelper.updateTable(insertString_PostProcessingCollaborators);
```

```
                }
                }

            }catch(Exception e) {
                    System.out.println("insertion in table post processing problem"+e.getMessage());
            }

            Hashtable qoscomponenttable = qoscomponent.getComponentQoS();
            Enumeration e1 = qoscomponenttable.keys();
            while (e1.hasMoreElements())
            {
                    qosfunction = (FunctionQoS)qoscomponenttable.get((String)e1.nextElement());
                    if (qosfunction !=null)
                    {
                    String functionname = qosfunction.getFunctionName();
                            Hashtable qosfunctiontable =  qosfunction.getFunctionQoS();
                    Enumeration e2 = qosfunctiontable.keys() ;
                    while(e2.hasMoreElements()) {
                                    try {
                                            String QoSParameter = (String) e2.nextElement();
                                    String value =
qosfunction.getFunctionQoS(component.getComponentName(), functionname, QoSParameter);
                                    String insertString_QoS = "INSERT INTO
"+componentTableName+"CompFuncQoS (" +
                                            "Id, ComponentName, SystemName, FunctionName,
QoSParameter, Value)" +

                                            "VALUES('" +component.getID() + "' , '" +
                                            component.getComponentName() + "' , '" +
                                            component.getSystemName() + "' , '" +
                                            functionname + "' , '" +
                                            QoSParameter + "' , '" +
                                            value + "')";

                                            sqlHelper.updateTable(insertString_QoS);
                                    }catch(Exception e){
                                            System.out.println("Error in updating tables
"+e.getMessage());
                                    }
                    }//End of inner while
                    }//End of if
            } //End of first while
        } //End of if
} //End of method
}//End of Class
```

Component.java

```java
import java.util.*;
import java.io.*;

/**
 * This class represents a component
 *
 * @author Zhisheng Huang
 * @date January 2003
 * @version 1.0
```

```
 */
abstract public class Component implements Serializable
{
   private String componentName = "";
   private String subcase = "";
   private String domainName = "";
   private String systemName = "";
   private String description = "";

   private String id = "";   //host id, seems not necessary in abastract component
   private String version = "";
   private String author = "";
   private String date = "";
   private String validity = "";
   private String atomicity = "Yes";
   private String registration = "";
   private String model = "";

   private String purpose = "";    //describe the function of the component
   private String[] algorithms = null;
   private String complexity = "";
   private String[] requiredInterfaces = null;
   private String[] providedInterfaces = null;
   private String[] technologies = null;
   private String[] expectedResources = null;
   private String[] designPatterns = null;
   private String[] knownUsages = null;
   private String[] aliases = null;

   private String[] preProcessingCollaborators = null;
   private String[] postProcessingCollaborators = null;

   private String mobility = "No";
   private String security = "";
   private String faultTolerance = "";

   private String[] qosMetrics = null;
   private String qosLevel = "";
   private String cost = "";
   private String qualityLevel = "";

   private ComponentQoS componentQoS;

   public void setComponentName (String componentName){this.componentName = componentName;}
   public void setSubcase(String subcase){this.subcase = subcase;}
   public void setDomainName (String domainName){this.domainName = domainName;}
   public void setSystemName(String systemName){this.systemName = systemName;}
   public void setDescription (String description){this.description = description;}

   public void setID (String id)  {this.id = id;}//host id, seems not necessary in abastract component
   public void setVersion (String version){this.version = version;}
   public void setAuthor (String author){this.author = author;}
   public void setDate (String date){this.date = date;}
   public void setValidity (String validity){this.validity =validity;}
   public void setAtomicity(String atomicity) {this.atomicity = atomicity;}
   public void setRegistration (String registration){this.registration = registration;}
```

```java
   public void setModel (String model){this.model = model;}

   public void setPurpose (String purpose) {this.purpose = purpose;} //describe the function of the
component
   public void setAlgorithms (String[] algorithms) {this.algorithms = algorithms;}
   public void setComplexity (String complexity){this.complexity = complexity;}
   public void setRequiredInterfaces (String[] interfaces) {this.requiredInterfaces = interfaces;}
   public void setProvidedInterfaces (String[] interfaces) {this.providedInterfaces = interfaces;}
   public void setTechnologies (String[] technologies){this.technologies = technologies;}
   public void setExpectedResources (String[] expectedResources){this.expectedResources =
expectedResources;}
   public void setDesignPatterns (String[] designPatterns){this.designPatterns = designPatterns;}
   public void setKnownUsages (String[] unknownUsages) {this.knownUsages = knownUsages;}
   public void setAliases (String[] aliases) {this.aliases = aliases;}

   public void setPreProcessingCollaborators (String[] collaborators) {this.preProcessingCollaborators =
collaborators;}
   public void setPostProcessingCollaborators (String[] collaborators) {this.postProcessingCollaborators =
collaborators;}

   public void setMobility (String mobility) {this.mobility = mobility;}
   public void setSecurity (String security) {this.security = security;}
   public void setFaultTolerance (String faultTolerance){this.faultTolerance = faultTolerance;}

   public void setQoSMetrics (String[] qosMetrics){this.qosMetrics = qosMetrics;}
   public void setQoSLevel (String qosLevel){this.qosLevel = qosLevel;}
   public void setCost (String cost){this.cost = cost;}
   public void setQualityLevel (String qualityLevel){this.qualityLevel = qualityLevel;}

   public void setComponentQoS(ComponentQoS componentQoS){this.componentQoS =
componentQoS;}

   public String getComponentName(){ return componentName;}
   public String getSubcase(){ return subcase;}
   public String getDomainName(){ return domainName;}
   public String getSystemName() {return systemName;}
   public String getDescription(){ return description;}

   public String getID(){ return id;}   //host id, seems not necessary in abastract component
   public String getVersion(){ return version;}
   public String getAuthor(){ return author;}
   public String getDate(){ return date;}
   public String getValidity(){ return validity;}
   public String getAtomicity () {return atomicity;}
   public String getRegistration(){ return registration;}
   public String getModel(){ return model;}

   public String getPurpose(){ return purpose;}    //describe the function of the component
   public String[] getAlgorithms(){ return algorithms;}
   public String getComplexity(){ return complexity;}
   public String[] getRequiredInterfaces () {return requiredInterfaces;}
   public String[] getProvidedInterfaces() {return providedInterfaces;}
   public String[] getTechnologies(){ return technologies;}
   public String[] getExpectedResources(){ return expectedResources;}
   public String[] getDesignPatterns(){ return designPatterns;}
   public String[] getKnownUsages(){ return knownUsages;}
```

```
   public String[] getAliases(){ return aliases;}

   public String[] getPreProcessingCollaborators() {return preProcessingCollaborators;}
   public String[] getPostProcessingCollaborators() {return postProcessingCollaborators;}

   public String getMobility (){return mobility;}
   public String getSecurity(){ return security;}
   public String getFaultTolerance(){ return faultTolerance;}

   public String[] getQoSMetrics() {return qosMetrics;}
   public String getQoSLevel(){ return qosLevel;}
   public String getCost(){ return cost;}
   public String getQualityLevel(){ return qualityLevel;}

   public ComponentQoS getComponentQoS(){return componentQoS;}

   public String toString()
   {
      return domainName + "/" + systemName + "/" + componentName;
   }
}
```

ComponentQoS.java

```
import java.util.*;
import java.io.*;

/**
 * This class represents the QoS for a component. A component is identified
 * by the system name and component name. The FunctionQoS for each method of
 * the component is stored in a Hashtable. The keys for the Hashtable are
 * function names of the component.
 *
 * @author Zhisheng Huang
 * @date January 2003
 * @version 1.0
 */
public class ComponentQoS implements Serializable
{
   private Hashtable componentQoS;
   private String componentName;
   private String systemName;

   /**
    * Constructor.
    *
    * @param systemName Name of the system.
    * @param componentName Name of the component.
    */
   public ComponentQoS(String systemName, String componentName)
   {
      this.componentName = componentName;
      this.systemName = systemName;
      componentQoS = new Hashtable();
   }
```

```
/**
 * This method add a FunctionQoS of a method associated with the component.
 * The method checks against the component name to ensure integrity. If the FunctionQoS
 * in the argument does not have the expected component name, this method does nothing.
 */
public void addFunctionQoS(FunctionQoS functionQoS)
{
   if(componentName.equals(functionQoS.getComponentName()))
   {
     String key = functionQoS.getFunctionName();
     componentQoS.put(key, functionQoS);
   }
}


/**
 * This method get a FunctionQoS for a specified function in the argument.
 * @param componentName Name of a component. For error checking.
 * @param functionName Name of a function.
 * @return FunctionQoS If the function specified in the argument exist
 *         return the corresponding FunctionQoS, else return null.
 */
public FunctionQoS getFunctionQoS(String componentName, String functionName)
{
   if(componentName.equals(this.componentName))
   {
     return (FunctionQoS)componentQoS.get(functionName);
   }
   else
   {
     return null;
   }
}


/**
 * This method returns the FunctionQoS of all the functions in the component
 * in a Hashtable.
 */
public Hashtable getComponentQoS()
{
   return (Hashtable)componentQoS.clone();
}


/**
 * This method returns the system name.
 */
public String getSystemName()
{
   return systemName;
}


/**
 * This method returns the component name.
 */
public String getComponentName()
{
   return componentName;
```

```
      }
}
```

ComponentSelectionAgent.java

```java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;
import java.util.*;
import java.io.*;
import java.lang.*;

import simpleCom.*;
import de.ikv.grasshopper.communication.GrasshopperAddress;
import de.ikv.grasshopper.communication.ExternalCommService;
import de.ikv.grasshopper.communication.ProxyGenerator;
import de.ikv.grasshopper.agency.IAgentSystem;
import de.ikv.grasshopper.agency.IRegionRegistration;
import de.ikv.grasshopper.agency.PlaceAlreadyExistsException;
import de.ikv.grasshopper.util.SearchFilter;
import de.ikv.grasshopper.type.AgentSystemInfo;
import de.ikv.grasshopper.type.AgentInfo;

/**
 * Insert the type's description here.
 * Creation date: (05/15/2003 11:50:10 AM)
 * @author: Jayasree Gandhamaneni
 */


public class ComponentSelectionAgent implements Runnable {

        private String clientLocation = null;
        private String agentCodebase = null;
        private GrasshopperAddress regionAddr = null;
        private ArrayList hhList = null;
        private QueryBean queryBean = null;
        private String queryID = null;

        /**
        * The ComponentSelectionAgent Constructor.
        */
        public ComponentSelectionAgent(
                ArrayList hhLocList,
                QueryBean querybean,
                String clientLoc,
                String registryAddress,
                String qID) {

                try {

                        System.setSecurityManager(new RMISecurityManager());
                        hhList = hhLocList;
                        clientLocation = clientLoc;
                        queryBean = querybean;
```

```
                                 regionAddr = new GrasshopperAddress(registryAddress);
                                 agentCodebase = "file:/c:/murds";
                                 queryID = qID;
                    } catch (Exception e){
                                 System.out.println("Exception in the constructor of DiscoveryAgent "+e);
                    }
        }

        public void run() {

                    Thread CurrentThread = Thread.currentThread();

                    try {
                                 System.out.println("QueryManager received query with "+ queryID +" from
client "+clientLocation);
                                 System.out.println("QueryManager obtained a list of "+hhList.size()+"
headhunters from DSM");
                                 createMobileAgent();
                    } catch (Exception e)
                    {
                                 System.out.println("Exception in the run() of DiscoveryAgent class
"+e.getMessage());
                    }
        }

        /**
        * Creates a QMAgent.
        */
        private void createMobileAgent()    {

                    try {
                                 IRegionRegistration regionProxy = (IRegionRegistration)

        ProxyGenerator.newInstance(IRegionRegistration.class,
                                                      regionAddr.generateRegionId(),
                                                      regionAddr);

                                 AgentSystemInfo[] agentSystemInfo = regionProxy.listAgencies(new
SearchFilter());

                                 if (agentSystemInfo.length > 0)
                                 {

                                            ArrayList agencyList=new ArrayList();
                                            for (int i=0; i<agentSystemInfo.length; i++)
                                            {
                                                        System.out.println( (i+1) + ". " +
agentSystemInfo[i].getLocation());
                                                        agencyList.add(agentSystemInfo[i].getLocation());
                                            }

                                            Hashtable agencyHHTable = new Hashtable();
                                            agencyHHTable = agencyHhIpMapping(hhList, agencyList);

                                            GrasshopperAddress address;
                                            Enumeration e = agencyHHTable.keys();
```

```
                                   if(e.hasMoreElements())
                                   {
                                           address = (GrasshopperAddress)e.nextElement();
                                           System.out.println("Contacting agency '" +address + "'.");

                                           String serverAddresses[] =
regionProxy.lookupCommunicationServer(address.generateAgentSystemId());

                                           GrasshopperAddress agencyAddress = new
GrasshopperAddress(serverAddresses[0]);
                                           IAgentSystem agencyProxy = (IAgentSystem)

      ProxyGenerator.newInstance(IAgentSystem.class,
                                                       agencyAddress.generateAgentSystemId(),
                                                       agencyAddress);

                                           Object agentCreationArgs[] = new Object[5];
                                           agentCreationArgs[0]
=(Hashtable)agencyHHTable;//agencyList consists of objects of type GrasshopperAddress
                                           agentCreationArgs[1]=(String)clientLocation.toString();

      agentCreationArgs[2]=(GrasshopperAddress)agencyAddress;//agency where the mobile agent is to
be created initially
                                           agentCreationArgs[3]=(QueryBean)queryBean;
                                           agentCreationArgs[4]=(String)queryID;
                                           System.out.println("QMAgent moving to agency
"+agencyAddress);
                                           AgentInfo agentInfo = agencyProxy.createAgent("QMAgent",
agentCodebase, "", agentCreationArgs);
                                   }
                                   else
                                   {
                                           System.out.println("Nothing to do. Please start agencies where
Headhunters are running.");
                                   }
                           }

               } catch(Exception e) {
                       e.printStackTrace();
               }
       }

       /**
        * This method maps the Headhunter locations with respective agencies.
        */
       private Hashtable agencyHhIpMapping(ArrayList hhAddressList, ArrayList agencyAddresses)
       {
               Hashtable DomainIPMapping=new Hashtable();

               if(hhAddressList.size()>0 && agencyAddresses.size()> 0)
               {
                       ArrayList tempAgency = new ArrayList();
                       ArrayList tempHHAdd = new ArrayList();

                       for(int k=0;k<agencyAddresses.size();k++)
                       {
```

```
                                        int j=0;
                                        String IPAdd=agencyAddresses.get(k).toString();
                                        int start=0;
                                        int t=0;
                                        for (int i=0;i<IPAdd.length();i++)
                                        {
                                                if(IPAdd.charAt(i)=='/')
                                                {
                                                        t++;
                                                        if(t==2)
                                                                start=i+1;
                                                }
                                        else if(IPAdd.charAt(i)==':' && t>=2)
                {
                        j =i;
                                i = IPAdd.length();
                }
                                        }
                                        IPAdd = IPAdd.substring(start,j);
                                        tempAgency.add(IPAdd);
                }


                        for(int k=0;k<hhAddressList.size();k++)
                        {
                                int j=0;
                                String MachineName=(String)hhAddressList.get(k);
                                int start=0;
                                int t=0;
                                for (int i=0;i<MachineName.length();i++)
                                {
                                        if(MachineName.charAt(i)=='/')
                                        {
                                                t++;
                                                if(t==2)
                                                        start=i+1;
                                        }
                                else if(MachineName.charAt(i)==':' && t>=2)
                {
                        j =i;
                                i = MachineName.length();
                }
                                }
                                MachineName = MachineName.substring(start,j);
                                tempHHAdd.add(MachineName);

                        }


                        try {

                                for(int i=0;i<tempAgency.size();i++)
                                {
                                        String hostData[] = new String[2];
                                        InetAddress addr =
InetAddress.getByName((String)(tempAgency.get(i)));
```

```
                                          hostData[0] = addr.getHostName();
                                          hostData[1] = addr.getHostAddress();
                                          ArrayList tempHHList=new ArrayList();
                                          for(int l=0;l<tempHHAdd.size();l++)
                                          {
                                                  if(hostData[1].equals((String)tempHHAdd.get(l)))
                                                  {
                                                          tempHHList.add(hhAddressList.get(l));
                                                  }
                                          }

                                          if(tempHHList.size()>0)

        DomainIPMapping.put((GrasshopperAddress)agencyAddresses.get(i),(ArrayList)tempHHList);
                                  }

                          }catch (java.net.UnknownHostException e) {
                          }
                  }

                  return DomainIPMapping;
          }
}
```

ConcreteComponent.java

```
public class ConcreteComponent extends Component
{
  public ConcreteComponent(){}
}
```

DiscoveryAgent.java

```
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;
import java.util.*;
import java.io.*;
import java.lang.*;

import simpleCom.*;
import de.ikv.grasshopper.communication.GrasshopperAddress;
import de.ikv.grasshopper.communication.ExternalCommService;
import de.ikv.grasshopper.communication.ProxyGenerator;
import de.ikv.grasshopper.agency.IAgentSystem;
import de.ikv.grasshopper.agency.IRegionRegistration;
import de.ikv.grasshopper.agency.PlaceAlreadyExistsException;
import de.ikv.grasshopper.util.SearchFilter;
import de.ikv.grasshopper.type.AgentSystemInfo;
import de.ikv.grasshopper.type.AgentInfo;

/**
 * Insert the type's description here.
 * Creation date: (05/15/2002 11:50:10 AM)
 * @author: Jayasree Gandhamaneni
```

```
 */


public class DiscoveryAgent implements Runnable {

        private long discoveryTime = 0;
        private IDomainSecurityManager dsmanager = null;
        private String hhLocation = null;
        private String domain = null;
        private String agentCodebase = null;
        private GrasshopperAddress regionAddr = null;
        private String mobileAgentUserName = null;

        /**
        * The DiscoveryAgent Constructor.
        */
        public DiscoveryAgent(
                long dTime,
                String dsmLoc,
                String hhLoc,
                String hhDomain,
                String registryAddress,
                String maUserName) {

                try {

                        System.setSecurityManager(new RMISecurityManager());
                        dsmanager = (IDomainSecurityManager) Naming.lookup(dsmLoc);

                        hhLocation = hhLoc;
                        domain = hhDomain;
                        discoveryTime = dTime;
                        regionAddr = new GrasshopperAddress(registryAddress);
                        agentCodebase = "file:/c:/murds";
                        mobileAgentUserName = maUserName;

                } catch (Exception e){
                        System.out.println("Exception in the constructor of DiscoveryAgent "+e);
                }
        }

        public void run() {

                Thread CurrentThread = Thread.currentThread();

                try {
                        while(true)
                        {

                                CurrentThread.sleep(discoveryTime);
                                ArrayList arList =
dsmanager.getARListForDomain(hhLocation,domain);
                                if(arList.size()>0)
                                {
                                        System.out.println("----------------------");
                                        System.out.println("Headhunter "+hhLocation+" received a list
```

```
of "+arList.size()+" Active registries from DSM");
                                       System.out.println("Active Registries available are as
follows");

                                       for(int i=0;i<arList.size();i++)
                                               System.out.println(i+"."+arList.get(i));
                                       System.out.println("Headhunter "+hhLocation+" sending
HHAgent for component discovery");

                                       createMobileAgent(arList);

                               }
                               else
                               {
                                       System.out.println("Agent couldn't find any active registries to
visit");

                                       System.out.println("Start some active registries in the system");
                               }
                       }
               }catch(InterruptedException ie) {
                       System.out.println(ie.getMessage());
               }
               catch (Exception e)
               {
                       System.out.println("Exception in the run() of DiscoveryAgent class
"+e.getMessage());
               }
       }

       /**
        * Creates a HHAgent.
        */
       private void createMobileAgent(ArrayList arList) {

               try {
                       IRegionRegistration regionProxy = (IRegionRegistration)

       ProxyGenerator.newInstance(IRegionRegistration.class,
                                               regionAddr.generateRegionId(),
                                               regionAddr);

                       AgentSystemInfo[] agentSystemInfo = regionProxy.listAgencies(new
SearchFilter());

                       if (agentSystemInfo.length > 0)
                       {

                               ArrayList agencyList=new ArrayList();
                               for (int i=0; i<agentSystemInfo.length; i++)
                               {
                                       agencyList.add(agentSystemInfo[i].getLocation());
                               }

                               Hashtable agencyARTable = new Hashtable();
                               agencyARTable = agencyArIpMapping(arList, agencyList);

                               GrasshopperAddress address;
                               Enumeration e = agencyARTable.keys();
                               if(e.hasMoreElements())
```

```
                                    {
                                            address = (GrasshopperAddress)e.nextElement();

                                            //it may be possible that an agency can have more than one
server to recieve communication..
                                            //for Ex: one server with rmi protocol and the other with the
socket protocol

                                            String serverAddresses[] =
regionProxy.lookupCommunicationServer(address.generateAgentSystemId());

                                            GrasshopperAddress agencyAddress = new
GrasshopperAddress(serverAddresses[0]);

                                            IAgentSystem agencyProxy = (IAgentSystem)
        ProxyGenerator.newInstance(IAgentSystem.class,
                                                    agencyAddress.generateAgentSystemId(),
                                                    agencyAddress);


                                            Object agentCreationArgs[] = new Object[5];
                                            agentCreationArgs[0] = agencyARTable;
                                            agentCreationArgs[1] = (String)hhLocation.toString();
                                            agentCreationArgs[2] = (String)mobileAgentUserName;
                                            agentCreationArgs[3] = (GrasshopperAddress)agencyAddress;

                                            Long startTime = new Long((new java.util.Date()).getTime());
                                            agentCreationArgs[4] = startTime;
                                            System.out.println("HHAgent moving to agency
"+agencyAddress);
                                            AgentInfo agentInfo = agencyProxy.createAgent("HHAgent",
agentCodebase, "", agentCreationArgs);
                                    }
                                    else
                                    {
                                            System.out.println("Nothing to do. Please start agencies where
ActiveRegistries are running.");
                                    }
                            }

                } catch(Exception e){
                        e.printStackTrace();
                }
        }

        /**
        * This method maps the ActiveRegistry locations with respective agencies.
        */
        private Hashtable agencyArIpMapping(ArrayList arAddressList, ArrayList agencyAddresses)
        {
                Hashtable DomainIPMapping=new Hashtable();

                if(arAddressList.size()>0 && agencyAddresses.size()> 0)
                {
                        ArrayList tempAgency=new ArrayList();
```

```
                        ArrayList tempARAdd=new ArrayList();

                        for(int k=0;k<agencyAddresses.size();k++)
                        {
                                int j=0;
                                String IPAdd=agencyAddresses.get(k).toString();
                                int start=0;
                                int t=0;
                                for (int i=0;i<IPAdd.length();i++)
                                {
                                        if(IPAdd.charAt(i)=='/')
                                        {
                                                t++;
                                                if(t==2)
                                                        start=i+1;
                                        }
                                else if(IPAdd.charAt(i)==':' && t>=2)
        {
                j =i;
                                i = IPAdd.length();
        }
                                }
                                IPAdd = IPAdd.substring(start,j);
                                tempAgency.add(IPAdd);
                        }


                for(int k=0;k<arAddressList.size();k++)
                        {
                                int j=0;
                                String MachineName=(String)arAddressList.get(k);
                                int start=0;
                                int t=0;
                                for (int i=0;i<MachineName.length();i++)
                                {
                                        if(MachineName.charAt(i)=='/')
                                        {
                                                t++;
                                                if(t==2)
                                                        start=i+1;
                                        }
                                else if(MachineName.charAt(i)==':' && t>=2)
        {
                j =i;
                                i = MachineName.length();
        }
                                }
                                MachineName = MachineName.substring(start,j);
                                tempARAdd.add(MachineName);

                }


                try
                {
                        for(int i=0;i<tempAgency.size();i++)
```

```
                                    {
                                            String hostData[] = new String[2];
                                            InetAddress addr =
InetAddress.getByName((String)(tempAgency.get(i)));
                                            hostData[0] = addr.getHostName();
                                            hostData[1] = addr.getHostAddress();
                                            ArrayList tempARList=new ArrayList();
                                            for(int l=0;l<tempARAdd.size();l++)
                                            {
                                                    if(hostData[1].equals((String)tempARAdd.get(l)))
                                                    {
                                                            tempARList.add(arAddressList.get(l));
                                                    }
                                            }
                                            if(tempARList.size()>0)

        DomainIPMapping.put((GrasshopperAddress)agencyAddresses.get(i),(ArrayList)tempARList);
                                    }

                            }catch (java.net.UnknownHostException e) {
                            //hostData[0] = e.toString();
                            //hostData[1] = e.toString();
                            }
                    }

                    return DomainIPMapping;
            }
}
```

DomainSecurityManager.java

```java
import java.net.*;
import java.util.*;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

/**
 * The DomainSecurityManager class uses the PrincipalAvailabilityChecker
 * to periodically check the availability of all the active registries and
 * the headhunters registered with it. After checking the availability of
 * principals, the PrincipalAvailabilityChecker updates the list of currently
 * available Headhunters and active registries with the DSM.
 * Insert the type's description here.
 * Creation date: (05/15/2002 10:50:10 AM)
 * @author: Jayasree Gandhamaneni
 */

public class DomainSecurityManager extends UnicastRemoteObject implements IDomainSecurityManager
{

        private static Hashtable userdomainMapping = null;
        private static Hashtable mobileUserPrincipalMapping = null;
        private static Hashtable registeredHHTable = new Hashtable();
        private static Hashtable registeredHHAgentTable = new Hashtable();
        private static Hashtable availableHHTable = new Hashtable();
```

```java
        private static Hashtable registeredHHTimestampTable = new Hashtable();
        private static Hashtable registeredARTable = new Hashtable();
        private static Hashtable availableARTable = new Hashtable();
        private static Hashtable registeredARTimestampTable = new Hashtable();

        /**
        * The DomainSecurityManager Constructor.
        */
        public DomainSecurityManager(long cTime,String dsmLocation) throws RemoteException {
                super();
                try {

                        loadUserDomainMappingData();
                        loadMobileAgentInfo();
                } catch (DomainSecurityManagerException e) {
                        System.out.println(e);
                }
        }

        public static void main(String[] args) {

                long checkingTime = 60000;//60 secs
                String dsmLocation="//"+args[0]+":"+args[1]+"/DomainSecurityManager";
                try
                {
                        System.setSecurityManager(new RMISecurityManager());
                        Naming.rebind(dsmLocation, new DomainSecurityManager(checkingTime,
dsmLocation));
                        System.out.println("DomainSecurityManager " + dsmLocation + " is ready.");

                        //Start a thread to check availability of registered ARs and HHs in the network
                        PrincipalAvailabilityChecker checker = new
PrincipalAvailabilityChecker(checkingTime,dsmLocation);
                        Thread checkingThread = new Thread(checker);
                        checkingThread.start();


                } catch (Exception e) {
                        System.out.println("DomainSecurityManager failed: " + e);
                }

        }

        /**
        * This method returns Headhunter list under a particular domain to the QM.
        */
        public java.util.ArrayList getHHListForDomain(String domainName)
                throws RemoteException {

                ArrayList hhList = new ArrayList();
                Enumeration e = registeredHHTable.keys();

                while (e.hasMoreElements()) {
                        String key = (String) e.nextElement();
                        if(availableHHTable.containsKey(key) &&
registeredHHTable.containsKey(key))
```

```java
                                    {
                                            String value = (String) registeredHHTable.get(key);
                                            if((value).equalsIgnoreCase(domainName))
                                    {
                                                    hhList.add(key);
                                                    //System.out.println("Headhunter "+key);
                                    }//endif
                                    }
                    }//end while
                    return hhList;
        }

        /**
        * This method returns ActiveRegistry list under a particular domain to the HH.
        */
        public java.util.ArrayList getARListForDomain(String headhunterLocation, String domainName)
                    throws RemoteException {
                    System.out.println("DSM Contacted by headhunter "+headhunterLocation+" for AR List
for : " + domainName + " Domain");

                    //System.out.println("DSM checking headhunter's validity ");

                    ArrayList arList = new ArrayList();
                    if(availableHHTable.containsKey(headhunterLocation) &&
((String)registeredHHTable.get(headhunterLocation)).equals(domainName))
                    {
                            //System.out.println("Headhunter "+headhunterLocation+" is a valid entity");
                            Enumeration e = registeredARTable.keys();

                            while (e.hasMoreElements())
                            {
                                    String key = (String) e.nextElement();
                                    if(availableARTable.containsKey(key))
                                    {
                                    String value = (String) registeredARTable.get(key);
                                            if((value).equalsIgnoreCase(domainName))
                                    {
                                                    arList.add(key);
                                    }//end if
                                    }//end if
                            }//end while
                    }
                    return arList;
        }

        /**
        * This method authenticates a HHAgent.
        */
        public String authenticateHHAgent(String hhLocation, String maUserName) {

                    boolean isAValidUser = false;
                    String serviceType = new String("");
                    if(availableHHTable.containsKey(hhLocation) &&
((String)registeredHHAgentTable.get(hhLocation)).equals(maUserName))
                    {
                            Random r= new Random();
```

```
                    int serviceSelector = r.nextInt(4)+1;
                    if(serviceSelector == 1)
                            serviceType = "PB";// PB=PremiumBusiness
                    else if(serviceSelector == 2)
                            serviceType = "PI"; // PI=PremiumIndividual
                    else if(serviceSelector == 3)
                            serviceType = "RB"; // RB=RegularBusiness
                    else if(serviceSelector == 4)
                            serviceType = "RI"; // PI=RegularIndividual
            }

            return serviceType;
        }

        /**
        * This method renews the ActiveRegistry state.
        */
        public void renewARState(String arLocation, String domain) {

                if(registeredARTable.containsKey(arLocation) &&
((String)registeredARTable.get(arLocation)).equals(domain))
                {
                        registeredARTimestampTable.put(arLocation, (new java.util.Date()));
                }
        }

        /**
        * This method renews the Headhunter state.
        */
        public void renewHHState(String hhLocation, String domain) {

                if(registeredHHTable.containsKey(hhLocation) &&
((String)registeredHHTable.get(hhLocation)).equals(domain))
                {
                        registeredHHTimestampTable.put(hhLocation, (new java.util.Date()));
                }
        }

        /**
        * This method returns ActiveRegistry time stamp table to the PrincipalAvailabilityChecker.
        */
        public Hashtable getARTimestampTable() {
                return registeredARTimestampTable;
        }

        /**
        * This method returns Headhunter time stamp table to the PrincipalAvailabilityChecker.
        */
        public Hashtable getHHTimestampTable() {
                return registeredHHTimestampTable;
        }

        /**
        * This method updates the freshness of the Headhunters and the ActiveRegistries.
        */
        public void receiveUpdatedTables(Hashtable arTable,Hashtable hhTable) {
```

```
                    availableARTable = arTable;
                    availableHHTable = hhTable;
            }

            /**
             * Authenticate principal against DSM_Repository.
             */
            private static boolean retrieveUser(
                    String userType,
                    String userName,
                    String password,
                    String domain)
                    throws DomainSecurityManagerException {

                    // load user from database
                    boolean userExists =
                            DSMRepositoryHelper.authenticateUser(userType, userName, password,
domain);

                    // if not found, throw exception
                    if (!userExists) {
                            throw new DomainSecurityManagerException(
                                    "User " + userName + " failed authentication.",
                                    null);
                    }

                    System.out.println("DSM authenticated " + userName);

                    return userExists;
            }

            /**
             * Remote method called by HH/AR.
             */
            public boolean authenticationService(
                    String userType,
                    String userName,
                    String password,
                    String location,
                    String domain)
                    throws RemoteException {

                    boolean isUserAuthenticated=false;

                    try
                    {
                            if(retrieveUser(userType, userName, password, domain)==true)
                            {
                                    isUserAuthenticated=true;
                                    if(userType.equals("Headhunter"))
                                    {
                                            //System.out.println("Headhunter "+ userName +"
authenticated by DSM at location"+ location);
                                            registeredHHTable.put(location,domain);
                                            availableHHTable.put(location,domain);
                                            //System.out.println("size of mobileUserPrincipalMapping "+
```

```
mobileUserPrincipalMapping.size());
                                        if(mobileUserPrincipalMapping.containsKey(userName))
                                        {
                                                //System.out.println("trying to get agent for
"+userName);
                                                String mobileAgent =
(String)mobileUserPrincipalMapping.get(userName);
                                                registeredHHAgentTable.put(location,mobileAgent);
                                                System.out.println("Mobile agent assocaited with
Headhunter at "+location+" is "+mobileAgent);
                                        }
                                }
                                else if(userType.equals("Registry"))
                                {
                                        registeredARTable.put(location,domain);
                                        availableARTable.put(location,(new java.util.Date()));
                                        //System.out.println("Active Registry "+ userName +"
authenticated by DSM at location"+ location);
                                }
                        }

                } catch (Exception e) {
                        System.out.println(e);
                }
                return isUserAuthenticated;
        }

        /**
        * Remote Method called by a HH to get the agent information.
        */
        public String getMobileAgentInfo(String headhunterLocation) {

                String mobileAgentName =new String();
                if(registeredHHAgentTable.containsKey(headhunterLocation))
                {
                        mobileAgentName = (String)registeredHHAgentTable.get(headhunterLocation);
                }
                return mobileAgentName;
        }

        /**
        * Method to load mobile agent and it's associated principal information from DSM_Repository
        */
        private static void loadMobileAgentInfo()throws DomainSecurityManagerException {

                try
                {
                        if (mobileUserPrincipalMapping == null)
                        {
                                mobileUserPrincipalMapping =
DSMRepositoryHelper.loadMobileUserPrincipalMapping();
                        }
                }
                catch (Exception e)
                {
                        throw new DomainSecurityManagerException("Error in loadQMAgentData
```

```
method", e);
                }
        }

        /**
        * Method to load users and their associated domain information from DSM_Repository
        */
        private static void loadUserDomainMappingData() throws DomainSecurityManagerException {

                try {
                        if (userdomainMapping == null) {
                                // initialize the jdbc helper class
                                DSMRepositoryHelper.initialize();
                                userdomainMapping =
DSMRepositoryHelper.loadUserDomainMapping();
                        }

                } catch (Exception e) {
                        throw new DomainSecurityManagerException("Error in init method", e);
                }
        }
}//end of DomainSecurityManager
```

DomainSecurityManagerException.java

```
public class DomainSecurityManagerException extends Exception
{

        private String message = null;
        private Exception exception = null;

        public DomainSecurityManagerException(String smessage, Exception ex)
        {
                // store the passed in values as class variables
                message = smessage;
                exception = ex;
        }

        public String getMessage()
        {
                // return the message to the user
                return message;
        }

        public void printStackTrace()
        {
                // output the message & print the StackTrace
                System.out.println( message);
                exception.printStackTrace();
        }
}
```

DSMRepositoryHelper.java

```
import java.sql.*;
import java.util.*;
```

```java
/**
 * This class performs functions associated with
 * accessing the DSM_Repository to retrieve
 * user-domain mappings and for user authentication.
 * Creation date: (05/15/2003 1:30:45 PM)
 * @author: Jayasree Gandhamaneni
 */
public class DSMRepositoryHelper {

private static SQLHelper sqlHelper = null;

/**
 * Initialize SQLHelper
 */
public static void initialize() {
        try {
                sqlHelper = new SQLHelper();

        } catch (Exception e) {
                System.out.println(e);
        }

}

/**
 * authenticate principal against DB.
 */

public static boolean authenticateUser(
        String sUserType,
        String sUserName,
        String sPassword,
        String sDomain) {
        boolean isAuthenticated = false;

        try {
                String sUserQuery =
                        "SELECT Users.UserName,Users.Password,Users.UserType,"+
                        +"Permissions.PermissionName From Users,Permissions,User_Permission_Xref
"
                                + "WHERE ( ( Users.UserName = '"
                                + sUserName
                                + "') AND ( Users.Password = '"
                                + sPassword
                                + "' ) AND ( Users.UserType = '"
                                + sUserType
                                + "' ) AND ( Permissions.PermissionName= '"
                                + sDomain
                                + "' ) AND (User_Permission_Xref.PermissionID =
Permissions.PermissionID)"+
                                + " AND (User_Permission_Xref.UserID = Users.UserID ) )";

                ResultSet resultSet = sqlHelper.executeQuery(sUserQuery);
```

```java
                            if (!resultSet.next()) {
                                    // the database has no results - therefore the user is not authenticated
                                    return false;
                            }

                            // retrieve the resultset
                            String sResult = resultSet.getString(1);

                            // if the username is returned,
                            if (sResult != null) {
                                    // the user has been successfully authenticated
                                    System.out.println("data available is "+sResult);
                                    isAuthenticated = sUserName.equals(sResult);
                            }

                } catch (SQLException sqlE) {
                            sqlE.printStackTrace();
                            return false;
                } catch (Exception e) {
                            e.printStackTrace();
                            return false;
                }

                return isAuthenticated;
}

/**
 * Build hashtable from resultset
 */
private static Hashtable getFeaturesFromResultSet(ResultSet resultSet)
                throws SQLException {

                Hashtable hFeatures = new Hashtable();

                String sFeatureName = null;
                String sFeatureValue = null;

                while (resultSet.next()) {
                            sFeatureName = resultSet.getString(1);
                            //System.out.println("key "+ sFeatureName);
                            sFeatureValue = resultSet.getString(2);
                            //System.out.println("value "+ sFeatureValue);

                            hFeatures.put(sFeatureName, sFeatureValue);
                }

                return hFeatures;
}

/**
 * Load from DomainList and Permission tables.
 */
public static Hashtable loadDomainList() {
                Hashtable hFeatures = null;

                String sDomainListQuery =
```

```
                            "SELECT DomainList.DomainAddress, Permissions.PermissionName"
                            + "From DomainList, Permissions  "
                            + " WHERE (DomainList.DomainID = Permissions.PermissionID)";
            try {
                    // execute the Query
                    ResultSet resultSet = sqlHelper.executeQuery(sDomainListQuery);
                    hFeatures = getFeaturesFromResultSet(resultSet);

            } catch (SQLException sqlE) {
                    sqlE.printStackTrace();
                    return null;
            } catch (Exception e) {
                    e.printStackTrace();
                    return null;
            }

            return hFeatures;
}

/**
 * Load from Users and Permissions.
public static Hashtable loadUserDomainMapping() {
            Hashtable hFeatures = null;

            String sUserDomainQuery =
                    "SELECT Users.UserName, Permissions.PermissionName "
                            + "FROM Users, Permissions, User_Permission_Xref "
                            + " WHERE ( "
                            + "(User_Permission_Xref.PermissionID = Permissions.PermissionID) AND "
                            + "(User_Permission_Xref.UserID = Users.UserID ) )";

            try {
                    // execute the Query
                    ResultSet resultSet = sqlHelper.executeQuery(sUserDomainQuery);
                    hFeatures = getFeaturesFromResultSet(resultSet);

            } catch (SQLException sqlE) {
                    sqlE.printStackTrace();
                    return null;
            } catch (Exception e) {
                    e.printStackTrace();
                    return null;
            }

            return hFeatures;
}

            /*Method added for getting agent data from database
              Added on 05/22/2003 by Jayasree */
            /**
             * Load mobile agent data from Mobile_Users.
             */
            public static Hashtable loadMobileUserPrincipalMapping()
            {
                    Hashtable hFeatures = null;
```

```java
                String sMobileUserPrincipalQuery = "SELECT PrincipalName,MobileUserName"
                                                        + " FROM Mobile_Users where
PrincipalType='Headhunter'";

                try
                {
                        // execute the Query
                        ResultSet resultSet = sqlHelper.executeQuery(sMobileUserPrincipalQuery);
                        hFeatures = getFeaturesFromResultSet(resultSet);

                }
                catch (SQLException sqlE)
                {
                        sqlE.printStackTrace();
                        return null;
                }
                catch (Exception e)
                {
                        e.printStackTrace();
                        return null;
                }

                return hFeatures;
        }

        /**
         * Load from Permissions table.
         */
        public static ArrayList getListOfDomains()
        {
                String sListofDomainsQuery = "SELECT PermissionName From Permissions";

                try
                {
                        // execute the Query
                        ResultSet resultSet = sqlHelper.executeQuery(sListofDomainsQuery);
                        // position to first record
                        boolean moreRecords = resultSet.next();
                        // If there are no records, display a message
                        if (!moreRecords) {
                                return null;
                        } else {
                                ArrayList listOfDomains = new ArrayList();
                                do {
                                        String domain = resultSet.getString("PermissionName");
                                        listOfDomains.add(domain);
                                } while (resultSet.next());
                                return listOfDomains;
                        } //end else

                } catch (SQLException sqlE) {
                        sqlE.printStackTrace();
                        return null;
                } catch (Exception e) {
                        e.printStackTrace();
                        return null;
```

```
                }
            }
}
```

ExtComServCreator.java

```java
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;
import java.net.*;

import de.ikv.grasshopper.communication.GrasshopperAddress;
import de.ikv.grasshopper.communication.ProxyGenerator;
import de.ikv.grasshopper.communication.ExternalCommService;
import de.ikv.grasshopper.type.Identifier;
import de.ikv.grasshopper.agency.AgentCreationFailedException;
import de.ikv.grasshopper.agency.IAgentSystem;
import de.ikv.grasshopper.agency.IRegionRegistration;
import de.ikv.grasshopper.util.SearchFilter;
import de.ikv.grasshopper.type.AgentSystemInfo;
import de.ikv.grasshopper.type.AgentInfo;




public class ExtComServCreator extends UnicastRemoteObject implements IExtComServCreator {
        ExternalCommService commService;

        /**
         * The ExtComServCreator Constructor.
         */
        public ExtComServCreator(String serviceHostAddress, String dsmLocation)throws
RemoteException {

                super();
                createExtComServ(serviceHostAddress,dsmLocation);
        }

        /**
         * Method to create external communication service.
         */
        private void createExtComServ(String serviceHostAddress,String dsmLocation)
        {
                try
                {

                        GrasshopperAddress commServiceAddress = new
GrasshopperAddress("rmi://"+serviceHostAddress+":3000/ComSer");
                ServerObject serverObject = new ServerObject(dsmLocation);

                        commService = new ExternalCommService(serverObject);
                        commService.start();
                commService.startReceiver(commServiceAddress);
        }
        catch(Exception e)
        {
                        System.out.println("In the createExtComServ() of class ExtComServCreator" );
```

```
                                    e.printStackTrace();
            }
    }

        /**
         * Method to close external communication service.
         */
        public void closeCommService()throws RemoteException
        {
                System.out.println("ExtComServCreator stopping communication service.");
                commService.stop();
        }

        public static void main(String args[])
        {
                System.setSecurityManager(new RMISecurityManager());
                try
                {
                        String bindingName = "//"+args[0]+":"+args[1]+"/ECSC";
                        String extComServiceAdd = args[0];
                        String dsmLocation = "//"+args[2]+":"+args[3]+"/DomainSecurityManager";
                        ExtComServCreator ecsc = new
ExtComServCreator(extComServiceAdd,dsmLocation);
                        Naming.rebind(bindingName,ecsc);
                        System.out.println("External Communication Service Ready!");
                }
                catch (Exception e)
                {
                        System.out.println("Exception: " + e.getMessage());
                        e.printStackTrace();
                }
        }
}
```

FunctionBean.java

```
import java.util.*;
import java.io.*;

/**
 * Insert the type's description here.
 * Creation date: (11/15/2001 11:50:10 AM)
 * @author: Nanditha Nayani
 */

public class FunctionBean implements Serializable{
        private java.lang.String functionName = "";
        private java.lang.String syntacticContract = "";
/**
 * FunctionBean constructor comment.
 */
public FunctionBean() {
        super();
}
/**
 * Insert the method's description here.
```

```
 * Creation date: (11/15/2001 11:57:03 AM)
 * @param resultSet java.sql.ResultSet
 * @exception java.lang.Exception The exception description.
 */
public void buildBean(java.sql.ResultSet resultSet) throws java.lang.Exception {

        id = resultSet.getString("id");
        functionName = resultSet.getString("function_name");
        syntacticContract = resultSet.getString("syntactic_contract");

}
/**
 * Insert the method's description here.
 * Creation date: (11/15/2001 11:50:52 AM)
 * @return java.lang.String
 */
public java.lang.String getFunctionName() {
        return functionName;

}
/**
 * Insert the method's description here.
 * Creation date: (11/15/2001 11:51:10 AM)
 * @return java.lang.String
 */
public java.lang.String getSyntacticContract() {
        return syntacticContract;

}
/**
 * Insert the method's description here.
 * Creation date: (11/15/2001 11:50:52 AM)
 * @param newFunctionName java.lang.String
 */
public void setFunctionName(java.lang.String newFunctionName) {
        functionName = newFunctionName;

}
/**
 * Insert the method's description here.
 * Creation date: (11/15/2001 11:51:10 AM)
 * @param newSyntacticContract java.lang.String
 */
public void setSyntacticContract(java.lang.String newSyntacticContract) {
        syntacticContract = newSyntacticContract;
}
        private java.lang.String id = "";
public java.lang.String getId() {
        return id;
}

public void setId(java.lang.String newId) {
        id = newId;
}public void persistBean(SQLHelper sqlHelper,String functionTableName) throws Exception {

        String functionUpdateString =
        "INSERT INTO "+functionTableName+" VALUES(" +
        "'" + id + "'," +
        "'" + functionName + "'," +
        "'" + syntacticContract + "')";
```

```
        sqlHelper.updateTable(functionUpdateString);
}}
```

FunctionQoS.java

```java
import java.util.*;
import java.io.*;

/**
 * This class represents the QoS for a function. A function is identified
 * by the component name and function name. The QoS values for corresponding
 * QoS parameters are stored in a Hashtable. The keys for the Hashtable are
 * the QoS parameters.
 *
 * @author Zhisheng Huang
 * @date January 2003
 * @version 1.0
 */
public class FunctionQoS implements Serializable
{
   private String componentName;
   private String functionName;
   private Hashtable functionQoSTable;

   /**
    * Constructor.
    *
    * @param componentName Name of the component.
    * @param functionName Name of the function.
    */
   public FunctionQoS(String componentName, String functionName)
   {
      this.componentName = componentName;
      this.functionName = functionName;
      functionQoSTable = new Hashtable();
   }

   /**
    * This method adds a pair of QoS parameter and its value.
    *
    * @param componentName Name of the component. It is used for error checking.
    * @param functionName Name of the function. It is used for error checking.
    * @param QoSParameter Name of the QoS parameter.
    * @param value Vaue for the QoS parameter shown as the third argument of this method.
    */
   public void addFunctionQoS(String componentName, String functionName, String QoSParameter,
String value)
   {
      if(this.componentName.equals(componentName) && this.functionName.equals(functionName))
      {
         functionQoSTable.put(QoSParameter, value);
      }
   }

   /**
```

```
     * This method get the value for the QoS parameter specified as the third argument.
     * The first two arguments are for error checking purpose.
     */
    public String getFunctionQoS(String componentName, String functionName, String QoSParameter)
    {
      if(componentName.equals(this.componentName) && functionName.equals(this.functionName))
      {
        return (String)functionQoSTable.get(QoSParameter);
      }
      else
      {
        return null;
      }
    }

    /**
     * This method return all the QoS metrics for the function as a Hashtable.
     */
    public Hashtable getFunctionQoS()
    {
      return (Hashtable)functionQoSTable.clone();
    }

    /**
     * This method returns the name of the component.
     */
    public String getComponentName()
    {
      return componentName;
    }

    /**
     * This method returns the name of the function.
     */
    public String getFunctionName()
    {
      return functionName;
    }
}
```

Headhunter.java

```
import java.net.*;
import java.util.*;
import java.sql.*;

import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

import java.security.*;
import java.io.*;
import java.lang.*;

import simpleCom.*;
import de.ikv.grasshopper.communication.GrasshopperAddress;
```

```
import de.ikv.grasshopper.communication.ExternalCommService;
import de.ikv.grasshopper.communication.ProxyGenerator;
import de.ikv.grasshopper.agency.IAgentSystem;
import de.ikv.grasshopper.agency.IRegionRegistration;
import de.ikv.grasshopper.agency.PlaceAlreadyExistsException;
import de.ikv.grasshopper.util.SearchFilter;
import de.ikv.grasshopper.type.AgentSystemInfo;
import de.ikv.grasshopper.type.AgentInfo;

/**
 * This class implements the IHeadhunter interface.The Headhunter
 * class interacts with the DomainSecurityManager using RMI-JRMP.
 * The Headhunter class interacts with the ActiveRegistry services
 * through Grasshopper mobile agents for the purpose of obtaining
 * the component information as a Hashtable of componentBean objects.
 * The Headhunter class delegates the job of periodically updating
 * its availability to the HHStateRenewer thread. The Headhunter class
 * delegates the job of periodic component discovery to the DiscoveryAgent
 * thread. The Headhunter persists this information to the
 * Meta_Repository and uses the MetaRepositoryHelper for the purpose
 * of performing searches against the repository. The SQL query for
 * the searches is obtained from the QueryBean which is propagated
 * to the Headhunter by a mobile agent acting on behalf of the
 * QueryManager.
 * Insert the type's description here.
 * Creation date: (05/15/2003 11:50:10 AM)
 * @author: Jayasree Gandhamaneni
 */

public class Headhunter extends UnicastRemoteObject implements IHeadhunter
{
        private java.lang.String userType = "Headhunter";
        private Hashtable registryTable = new Hashtable();
        private Hashtable resultTable=null;
        private String componentTableName=null;
        private String hhLocation=null;

        private String registryAddress = null;
        private String agentCodebase = null;

        /**
         * Remote method called by the QMAgent to get components available with the HH.
         */
        public Hashtable performSearch(QueryBean querybean) throws RemoteException
        {
                Hashtable resultTable = new Hashtable();
                System.out.println("Processing Query request ");
                MetaRepositoryHelper srchEngine=new MetaRepositoryHelper(querybean);
                try
                {
                        long sTime =(new java.util.Date()).getTime();
                        resultTable = srchEngine.getSearchResultTable(componentTableName);
                        System.out.println("Total time taken to retrieve components from meta-rep is "+
                                        ((new java.util.Date()).getTime()-sTime));
                }catch(Exception e)
                {
```

```
                        System.out.println("Error in searching Local Metarepository"+ e.getMessage());
        }
        return resultTable;
}

/**
 * Method to create Meta_Repository.
 */
private void createMetaRepository(String userName) throws Exception
{
SQLHelper sqlEngine = new SQLHelper();
String dropUmmSpecs = "DROP TABLE "+userName+"UMMSpecification";
String dropAlgorithms = "DROP TABLE "+userName+"Algorithms";
String dropreqdinterface = "DROP TABLE "+userName+"RequiredInterfaces";
String dropProvidedInterfaces = "DROP TABLE "+userName+"ProvidedInterfaces";
String dropTechnologies = "DROP TABLE "+userName+"Technologies";
String dropExpectedResources = "DROP TABLE "+userName+"ExpectedResources";
String dropDesignPatterns = "DROP TABLE "+userName+"DesignPatterns";
String dropKnownUsages = "DROP TABLE "+userName+"KnownUsages";
String dropAliases = "DROP TABLE "+userName+"Aliases";
String dropPreProcessing = "DROP TABLE "+userName+"PreProcessing";
String dropPostProcessing = "DROP TABLE "+userName+"PostProcessing";
String dropCompFuncQoS = "DROP TABLE "+userName+"CompFuncQoS";
try {
 sqlEngine.updateTable(dropUmmSpecs);
}catch (Exception e){}
try {
 sqlEngine.updateTable(dropAlgorithms);
}catch (Exception e){}
try {
                sqlEngine.updateTable(dropreqdinterface);
}catch (Exception e){}
try {
                sqlEngine.updateTable(dropProvidedInterfaces);
        }catch (Exception e){}
try {
                sqlEngine.updateTable(dropTechnologies);
}catch (Exception e){}
try {
                sqlEngine.updateTable(dropExpectedResources);
}catch (Exception e){}
try {
                sqlEngine.updateTable(dropDesignPatterns);
        }catch (Exception e){}
        try {
                sqlEngine.updateTable(dropKnownUsages);
}catch (Exception e){}
try {
                sqlEngine.updateTable(dropAliases);
}catch (Exception e){}
try {
                sqlEngine.updateTable(dropPreProcessing);
}catch (Exception e){}
try {
                sqlEngine.updateTable(dropPostProcessing);
}catch (Exception e){}
```

```
    try {
                    sqlEngine.updateTable(dropCompFuncQoS);
    }catch (Exception e)
    {
            //System.out.println("Exception in dropping table"+e.getMessage());
    }
             String createUMMSpecification = "create table "+ userName + "UMMSpecification" +
             "(ComponentName VARCHAR(256), " +
             "SubCase VARCHAR(256), " +
             "DomainName VARCHAR(256), " +
             "SystemName  VARCHAR(256), " +
             "Description VARCHAR(2000), " +
             "Id VARCHAR(256) PRIMARY KEY, " +
             "Version VARCHAR(256), " +
             "Author VARCHAR(256), " +
             "CreatingDate VARCHAR(256), " +
             "Validity VARCHAR(256), " +
             "Atomicity VARCHAR(256), " +
             "Registration VARCHAR(256), " +
             "Model VARCHAR(256), " +
             "Purpose VARCHAR(2000), " +
             "Complexity VARCHAR(256), " +
             "Mobility VARCHAR(256), " +
             "Security VARCHAR(256), " +
             "FaultTolerance VARCHAR(256), " +
             "QosLevel VARCHAR(256), " +
             "Cost VARCHAR(256), " +
             "QualityLevel VARCHAR(256))";

    String createAlgorithms = "Create table "+ userName + "Algorithms" +
             "(Id VARCHAR(256) NOT NULL, " +
                     "ComponentName VARCHAR(256), " +
             "DomainName VARCHAR(256), " +
             "SystemName VARCHAR(256), " +
             "Algorithm VARCHAR(256) NOT NULL," +
             " PRIMARY KEY (Id, Algorithm))";

    String createRequiredInterfaces = "Create table "+ userName + "RequiredInterfaces" +
             "(Id VARCHAR(256) NOT NULL, " +
             "ComponentName VARCHAR(256), " +
             "DomainName VARCHAR(256), " +
             "SystemName VARCHAR(256), " +
             "Interface VARCHAR(256) NOT NULL, " +
                             " PRIMARY KEY (Id, Interface))";


    String createProvidedInterfaces = "create table "+ userName + "ProvidedInterfaces" +
             "(Id VARCHAR(256) NOT NULL, " +
             "ComponentName VARCHAR(256), " +
             "DomainName VARCHAR(256), " +
             "SystemName VARCHAR(256), " +
             "Interface VARCHAR(256) NOT NULL, " +
             " PRIMARY KEY (Id, Interface))";

    String createTechnologies = "create table "+ userName + "Technologies" +
             "(Id VARCHAR(256) NOT NULL, " +
```

```
                    "ComponentName VARCHAR(256), " +
                    "DomainName VARCHAR(256), " +
                    "SystemName VARCHAR(256), " +
                    "Technology VARCHAR(256) NOT NULL," +
                    " PRIMARY KEY (Id, Technology))";

        String createExpectedResources = "create table "+ userName + "ExpectedResources" +
                                "(Id VARCHAR(256) NOT NULL, " +
                    "ComponentName VARCHAR(256), " +
                    "DomainName VARCHAR(256), " +
                    "SystemName VARCHAR(256), " +
                    "ExpectedResource VARCHAR(256) NOT NULL," +
                    " PRIMARY KEY (Id, ExpectedResource))";

        String createDesignPatterns = "create table "+ userName + "DesignPatterns" +
                                "(Id VARCHAR(256) NOT NULL, " +
                    "ComponentName VARCHAR(256), " +
                    "DomainName VARCHAR(256), " +
                    "SystemName VARCHAR(256), " +
                    "Pattern VARCHAR(256) NOT NULL," +
                    " PRIMARY KEY (Id, Pattern))";

        String createKnownUsages = "create table "+ userName + "KnownUsages" +
                                "(Id VARCHAR(256) NOT NULL, " +
                    "ComponentName VARCHAR(256), " +
                    "DomainName VARCHAR(256), " +
                    "SystemName VARCHAR(256), " +
                    "Usage VARCHAR(256) NOT NULL," +
                    " PRIMARY KEY (Id, Usage))";

        String createAliases = "create table "+ userName + "Aliases" +
                    "(Id VARCHAR(256) NOT NULL, " +
                    "ComponentName VARCHAR(256), " +
                    "DomainName VARCHAR(256), " +
                    "SystemName VARCHAR(256), " +
                    "Alias VARCHAR(256) NOT NULL," +
                    " PRIMARY KEY (Id, Alias))";

        String createPreProcessingCollaborators = "create table "+ userName + "PreProcessing" +
                    "(Id VARCHAR(256) NOT NULL, " +
                    "ComponentName VARCHAR(256), " +
                    "DomainName VARCHAR(256), " +
                    "SystemName VARCHAR(256), " +
                    "Collaborator VARCHAR(256) NOT NULL," +
                    " PRIMARY KEY (Id, Collaborator))";

        String createPostProcessingCollaborators = "create table "+ userName + "PostProcessing" +
                    "(Id VARCHAR(256) NOT NULL, " +
                    "ComponentName VARCHAR(256), " +
                    "DomainName VARCHAR(256), " +
                    "SystemName VARCHAR(256), " +
                    "Collaborator VARCHAR(256) NOT NULL," +
                    " PRIMARY KEY (Id, Collaborator))";

        /* In the following table, the length of the columns of the tables have been changed, since Oracle
doesn't allow
```

```
        the length of the primary key to exceed 756 key length*/

String createqoscompfunc = "create table "+ userName + "CompFuncQoS" +
        "(Id VARCHAR(100) NOT NULL, " +
        "ComponentName VARCHAR(256), " +
        "SystemName VARCHAR(256), " +
        "FunctionName VARCHAR(100) NOT NULL, " +
        "QoSParameter VARCHAR(100) NOT NULL, " +
        "Value VARCHAR(100) NOT NULL, " +
        " PRIMARY KEY (Id, FunctionName, QoSParameter, Value))";

try{
     sqlEngine.updateTable(createUMMSpecification);
} catch(Exception e) {System.out.println("UMM Spec Table problem"+e.getMessage());}
try{
  sqlEngine.updateTable(createAlgorithms);
}catch(Exception e){System.out.println("Algo table problem");}
 try{
  sqlEngine.updateTable(createRequiredInterfaces);
}catch(Exception e){System.out.println("Required Interface table problem");}
 try{
  sqlEngine.updateTable(createProvidedInterfaces);
}catch(Exception e){System.out.println("Provided Interface table problem");}
     try{
  sqlEngine.updateTable(createTechnologies);
}catch(Exception e){System.out.println("Technology table problem");}
 try{
  sqlEngine.updateTable(createExpectedResources);
}catch(Exception e){System.out.println("Resources table problem");}
 try{
  sqlEngine.updateTable(createDesignPatterns);
}catch(Exception e){System.out.println("DesignPatterns table problem");}
 try{
  sqlEngine.updateTable(createKnownUsages);
}catch(Exception e){System.out.println("knownusages table problem");}
 try{
  sqlEngine.updateTable(createAliases);
}catch(Exception e){System.out.println("Aliases table problem");}
 try{
  sqlEngine.updateTable(createPreProcessingCollaborators);
}catch(Exception e){System.out.println("Preprocessing table problem");}
 try{
  sqlEngine.updateTable(createPostProcessingCollaborators);
}catch(Exception e){System.out.println("Postprocessing table problem");}
 try{
  sqlEngine.updateTable(createqoscompfunc);
}catch(Exception e){System.out.println(" Problem in qos update table: "+e.getMessage());}
        sqlEngine.shutDown();
  }


  public static void main(String[] args)
  {
          long renewalTime = 30000;
          String headhunterLocation = "//"+args[0]+":" +args[1]+"/HeadHunter";
          String dsmLocation = "//"+args[2]+":"+args[3]+"/DomainSecurityManager";
```

```
                    String regionRegistryAddress = "rmi://"+args[4]+":6020/MyRegion";
                    String domain = args[5];
                    String userName = args[6];
                    String password = args[7];
                    long dTime = 15000;

                    try {
                            System.setSecurityManager(new RMISecurityManager());
                            Naming.rebind(
                                    headhunterLocation,
                                    new Headhunter(
                                    renewalTime,
                                    dTime,
                                    userName,
                                    password,
                                    domain,
                                    headhunterLocation,
                                    dsmLocation,
                                    regionRegistryAddress));
                            System.out.println("Headhunter is ready.");
                    } catch (Exception e) {
                            System.out.println("HeadHunter failed: " + e);
                    }
            }

            /**
             * The Headhunter Constructor.
             */
            public Headhunter(
                    long rTime,
                    long dTime,
                    String userName,
                    String password,
                    String domain,
                    String headhunterLocation,
                    String dsmLocation,
                    String regionRegistryAddress)
                    throws RemoteException {

        System.out.println("\n Headhunter activated at " + headhunterLocation);
                    componentTableName=userName;
                    hhLocation=headhunterLocation;
                    boolean isAuthenticated=false;
                    String mobileAgentUserName = new String();
                    IDomainSecurityManager dsmanager =null;

                    try
                    {
                            System.out.println("Headhunter Contacting DSM for Authentication.");

                    //IDomainSecurityManager dsmanager =(IDomainSecurityManager)
        Naming.lookup(dsmLocation);
                    dsmanager =(IDomainSecurityManager) Naming.lookup(dsmLocation);

                    isAuthenticated = dsmanager.authenticationService(
                                    userType,
```

```
                                        userName,
                                        password,
                                        headhunterLocation,
                                       domain);

                        if(isAuthenticated)
                           {
                                mobileAgentUserName =
dsmanager.getMobileAgentInfo(headhunterLocation);
                                System.out.println("DSM returned agent info "+
mobileAgentUserName);

                                createMetaRepository(userName);
                                System.out.println("MetaRepository Created.");

                                //start a thread to periodically update headhunter's availability in the
system
                                HHStateRenewer hhStateRenewer = new
HHStateRenewer(rTime,dsmLocation,headhunterLocation,domain);
                                Thread renewerThread = new Thread(hhStateRenewer);
                                renewerThread.start();

                                DiscoveryAgent discoveryAgent = new DiscoveryAgent(dTime,

        dsmLocation,

        hhLocation,

        domain,

        regionRegistryAddress,

        mobileAgentUserName);
                                Thread discoveryThread = new Thread(discoveryAgent);
                                discoveryThread.start();
                           }
                        else
                           {
                                System.out.println("Headhunter is not a valid pricipal. Authentication
failed");
                                System.exit(0);
                           }
                }catch(Exception e)
                {
                        System.out.println(e.getMessage());
                }

        } //end of constructor

        /**
         * Method to populate Meta_Repository.
         */
        public void populateMetaRepository(Hashtable CompData,String resultType,long startTime,int
noOfMsgs)throws RemoteException
        {
                try
```

```
                    {
                            System.out.println("Headhunter "+hhLocation+" obtained registered services
data from HHAgent");
                            System.out.println("Number of components retrieved are "+CompData.size());
                            System.out.println("Number of messages taken by the HHAgent to discover
components are "+ (++noOfMsgs));
                    if(resultType.equals("final")) {
                            long elapsedTime = (new java.util.Date()).getTime() - startTime;
                            System.out.println("\nTotal time taken to discover components is
"+elapsedTime+" millisecs");
                            System.out.println("i.e. "+elapsedTime/1000+" seconds\n");
                    }

                    if(!CompData.isEmpty())
            {
                            SQLHelper sqlEngine = new SQLHelper();
                            Enumeration e = CompData.elements();

                            while (e.hasMoreElements())
            {
            ConcreteComponent component = new ConcreteComponent();
            component=(ConcreteComponent) e.nextElement();
                                    try
                                    {
                            buildpersist buildcomponent = new buildpersist();

    buildcomponent.persist(component,sqlEngine,componentTableName);
                                    }catch (Exception ex)
            {
                                            //ex.printStackTrace();
                                            //If the component already exists in the table, ignore
the error thrown by the database system
                                    }
                            }//end while

                            sqlEngine.shutDown();
            }
                    else
                            System.out.println("No components discovered");
                    System.out.println("-------------------");
            } catch (Exception ex) {
                    System.out.println("HH exception " + ex);
            }
        }
    }
}//end of HeadHunter
```

HHAgent.java
import de.ikv.grasshopper.communication.GrasshopperAddress;   import de.ikv.grasshopper.communication.ProxyGenerator;   import de.ikv.grasshopper.communication.ExternalCommService;   import de.ikv.grasshopper.type.Identifier;   import de.ikv.grasshopper.agency.AgentCreationFailedException;   import java.util.*;   import java.lang.*;

```
public class HHAgent extends de.ikv.grasshopper.agent.MobileAgent{

        private Hashtable agencyARMapping=null;
        private String originatingHHLocation=null;
        private String mobileAgentUserName=null;
        private GrasshopperAddress originatingAgencyAdd;
        private GrasshopperAddress targetAgencyAdd;
        private String ipAddress;

        private GrasshopperAddress commObjectAddress;
        private IServerObject serverObjectProxy;
        private Hashtable resultTable=new Hashtable();

        private int resultCounter = 0;
        private int randomResultResolver = 0;
        private boolean sendInterResults = false;
        private long startTime = 0;
        private int noOfMsgs = 0;
        public void init(Object[] args){
                log("*******************");
                log("In the init method of HHAgent");
        agencyARMapping=(Hashtable)args[0];
        originatingHHLocation=(String)args[1];
        mobileAgentUserName=(String)args[2];
                originatingAgencyAdd=(GrasshopperAddress)args[3];
                targetAgencyAdd=originatingAgencyAdd;
                ipAddress=(String)targetAgencyAdd.toString();
                startTime = ((Long)args[4]).longValue();

        commObjectAddress = new
GrasshopperAddress("rmi://"+parseIPAddress(ipAddress)+":3000/ComSer");
        serverObjectProxy = (IServerObject)ProxyGenerator.newInstance(IServerObject.class,
                commObjectAddress.generateAgentSystemId(),
                commObjectAddress);
                //code to send intermediate results to Headhunter

                int numberOfAgencies = agencyARMapping.size();
                if(numberOfAgencies>5) {
                        randomResultResolver = (int)(Math.random()*(int)(numberOfAgencies/2))+1;
                        if( randomResultResolver > 1)
                        {
                                sendInterResults = true;
                        }
                }

        }

        /**
        * The name of the agent.
        */
        public String getName() {
        return "HHAgent";
        }

        /**
        * Callback from the agent system after moving.
```

```
        */
        public void afterMove() {
        try
        {
                        ipAddress=(String)targetAgencyAdd.toString();
                        commObjectAddress = new
GrasshopperAddress("rmi://"+parseIPAddress(ipAddress)+":3000/ComSer");
                        serverObjectProxy =
(IServerObject)ProxyGenerator.newInstance(IServerObject.class,
                                commObjectAddress.generateAgentSystemId(),
                                commObjectAddress);
        } catch(Exception e)
        {
                        log("error in afterMove method");
                        log(e.getMessage());
        }
        }


        /**
   * The lifecycle of this agent.
   */
        public void live() {
                log("HHAgent arrived at "+targetAgencyAdd);
                noOfMsgs++;

        try
        {
                        ArrayList arList = new ArrayList();

                        if(agencyARMapping.size()> 0 &&
agencyARMapping.containsKey(targetAgencyAdd))
                        {
                                /* get the list of ActiveRegistries that are associated
                                with a particular agency on a particular machine */
                                arList=(ArrayList)(agencyARMapping.remove(targetAgencyAdd));
                        }

                        if(arList.size()==0)
                        {
                                log("This system doesn't have any active registries to visit");
                        }
                        else if(arList.size()>0)
                        {
                                resultCounter++;
                                log("Number of Active registries to be contacted on this host machine
are "+arList.size());
                                        while(!arList.isEmpty()) {
                                                //take one AR at a time
                                                String arAddress = (String)arList.remove(0);
                                                log("HHAgent contacting AR "+arAddress);
                                                //get the state of the AR
                                                log("Contacting AR "+arAddress);

                                                int state = serverObjectProxy.getARState(arAddress);
                                                log("AR state is "+state);
```

```
                                        String attributeType ="";
                                        String attributeValue ="";
                                        Hashtable tempCompData = new Hashtable();
                                        // if the AR is in state 1, just pass the HHlocation and the
HHAgent name and get the data
                                        //if the AR is in state 2, randomly pick a functional attribute
and ask the AR for comp. data
                                        if(state == 2)
                                        {
                                                Random r = new Random();
                                                int attribute = r.nextInt(3);

                                                if(attribute == 0)
                                                {
                                                        attributeType = "algorithm";
                                                        attributeValue = "JFC";
                                                }
                                                else if(attribute == 1)
                                                {
                                                        attributeType = "complexity";
                                                        attributeValue = "O(1)";
                                                }
                                                else if(attribute == 2)
                                                {
                                                        attributeType = "technology";
                                                        attributeValue = "Java RMI";
                                                }
                                                log("HHAgent requesting components based on
"+attributeType+" "+attributeValue);
                                        }

        tempCompData=serverObjectProxy.getCompDataFromAR(arAddress,originatingHHLocation,
mobileAgentUserName,attributeType,attributeValue);
                                        Enumeration e1=tempCompData.keys();
                                        if(!e1.hasMoreElements())
                                        {
                                                log("No components found!!!");
                                        }
                                        else
                                        {
                                                log("No. of Components retrieved are
"+tempCompData.size());

                                                log("Components retrieved are as follows");
                                        }

                                        while(e1.hasMoreElements()){
                                                String urlID=(String)e1.nextElement();
                                                ConcreteComponent component =
(ConcreteComponent)tempCompData.get(urlID);
                                                log("Component
"+component.getComponentName()+"  available at  "+urlID);
                                                //log("Component ID is "+component.getID());
                                                resultTable.put(urlID,component);

                                        }
                                }
```

```
                                        if( sendInterResults==true && (resultCounter%randomResultResolver
== 0))
                                        {
                                                //send results to Headhunter through serverObject and empty
the resultTable
                                                resultCounter = 0;

        serverObjectProxy.postCompDataToHH(originatingHHLocation,resultTable,"inter",startTime,noO
fMsgs);
                                                resultTable.clear();
                                        }
                                }
                                log("No. of agencies remained are "+agencyARMapping.size());
                                if(agencyARMapping.size()>0)
                                {
                                        Enumeration e=agencyARMapping.keys();
                                        GrasshopperAddress tempAgencyAdd=null;
                                        if(e.hasMoreElements())
                                        {
                                                tempAgencyAdd=(GrasshopperAddress)e.nextElement();
                                        }
                                        originatingAgencyAdd=targetAgencyAdd;
                                targetAgencyAdd=(GrasshopperAddress)(tempAgencyAdd);
                                log("HHAgent moving to agency "+targetAgencyAdd);
                                move(targetAgencyAdd);
                                }
                                else
                                {
                                        log("No more ActiveRegistries to visit");
                                        log("Finally sending the component data to Headhunter
"+originatingHHLocation);

        serverObjectProxy.postCompDataToHH(originatingHHLocation,resultTable,"final",startTime,noO
fMsgs);
                                        try
                                        {
                                                //log("trying to remove agent");
                                                remove();
                                                log("HHAgent removed from the system");
                                        }
                                        catch(Exception e) {
                                                log("failed to remove agent", e);
                                        }
                                }
                } catch (Exception e) {
                log("Migration failed. Exception = ", e);
                serverObjectProxy.printMessage("HHAgent couldn't move.");
                }
                }

                private String parseIPAddress(String IPAddress)
                {
                        int j=0;
                        int start=0;
                        int k=0;
                        for (int i=0;i<IPAddress.length();i++)
```

```
                {
                        if(IPAddress.charAt(i)=='/')
                        {
                                k++;
                                if(k==2)
                                        start=i+1;
                        }
        else if(IPAddress.charAt(i)==':' && k>=2)
        {
                j =i;
                i = IPAddress.length();
        }
                }
                return IPAddress.substring(start,j);
        }
}
```

HHStateRenewer.java

```java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;
import java.util.*;

/**
 * This class operates as a Thread which executes periodically
 * to update availability of a Headhunter with the
 * DomainSecurityManager.
 * Creation date: (06/15/2003 10:10:30 AM)
 * @author: Jayasree Gandhamaneni
 */

public class HHStateRenewer implements Runnable
{
        private long rTime = 0;
        private String hhLocation= "";
        private IDomainSecurityManager dsm = null;
        private String hhDomain = "";

        /**
         * The HHStateRenewer Constructor.
         */
        public HHStateRenewer(
                long renewalTime,
                String dsmLocation,
                String headhunterLocation,
                String domain ) {

                try {

                        System.out.println("HHStateRenewer thread starting headhunter state updation
with the DSM");
                        System.setSecurityManager(new RMISecurityManager());
                        dsm = (IDomainSecurityManager) Naming.lookup(dsmLocation);
                        rTime = renewalTime;
```

```
                              hhLocation= headhunterLocation;
                              hhDomain = domain;

                      }catch (Exception e){
                              System.out.println("Exception in the constructor of HeadhunterStateRenewer
"+e.getMessage());
                      }
        }

        public void run()
        {
                Thread CurrentThread = Thread.currentThread();

                try {
                        while(true)
                        {
                                dsm.renewHHState(hhLocation, hhDomain);
                                CurrentThread.sleep(rTime);
                        }
                }catch(InterruptedException ie) {
                        System.out.println(ie.getMessage());
                }
                catch (Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

IActiveRegistry.java

```java
import java.rmi.*;
import java.util.*;

public interface IActiveRegistry extends Remote
{
        public Hashtable getComponentData(String headhunterLocation, String
mobileAgentUserName, String attributeType, String attributeValue) throws
RemoteException;
        public int getState() throws RemoteException;
}
```

IComponent.java

```java
import java.rmi.*;

public interface IComponent extends Remote
{
         public String getUmmSpecURL() throws RemoteException;
}
```

IDomainSecurityManager.java

```java
import java.util.*;
import java.rmi.*;
```

```
/**
 * Insert the type's description here.
 * Creation date: (05/16/03 9:07:49 PM)
 * @author: Jayasree Gandhamaneni
 */
public interface IDomainSecurityManager extends java.rmi.Remote{

        public boolean authenticationService(String userType, String userName, String password, String
contactLocation, String domain) throws RemoteException;
        public String getMobileAgentInfo(String headhunterLocation) throws RemoteException;
        public void renewARState(String arLocation, String domain) throws RemoteException;
        public void renewHHState(String hhLocation, String domain) throws RemoteException;
        public void receiveUpdatedTables(Hashtable arTable,Hashtable hhTable) throws
RemoteException;
        public Hashtable getARTimestampTable()throws RemoteException;
        public Hashtable getHHTimestampTable()throws RemoteException;
        public ArrayList getARListForDomain(String headhunterLocation, String domainName)throws
RemoteException;
        public ArrayList getHHListForDomain(String domainName) throws RemoteException;
        public String authenticateHHAgent(String hhLocation, String maUserName)throws
RemoteException;
}
```

IExtComServCreator.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.*;

public interface IExtComServCreator extends java.rmi.Remote
{
        public void closeCommService() throws java.rmi.RemoteException;
}
```

IHeadhunter.java

```
import java.rmi.*;
import java.util.*;
import java.security.*;
import java.io.*;

public interface IHeadhunter extends Remote
{
        public Hashtable performSearch(QueryBean querybean) throws RemoteException;
        public void populateMetaRepository(Hashtable CompData,String resultType,long startTime,int
noOfMsgs)throws RemoteException;
}
```

IQueryManager.java

```
import java.rmi.*;
import java.util.*;


public interface IQueryManager extends Remote {
```

```
                public void getSearchResultTable(QueryBean qBean,String clientLocation,String qID) throws
RemoteException;
}
```

IServerObject.java

```
import java.util.*;
import de.ikv.grasshopper.communication.GrasshopperAddress;

public interface IServerObject {
        public void printMessage(String msg);
        public void postCompDataToHH(String hhLocation,Hashtable resultTable, String resultType,long
startTime,int noOfMsgs);
        public Hashtable getCompDataFromAR(String arAddress, String headhunterLocation, String
mobileAgentUserName,String attributeType, String attributeValue);
        public Hashtable getCompDataFromHH(ArrayList hhList, QueryBean queryBean);
        public void postCompDataToClient(String clientLocation,Hashtable resultTable,String
resultType, String qID);
        public int getARState(String arAddress);
}
```

IURDS_Proxy.java

```
import java.rmi.*;
import java.util.*;

/**
 * This interface defines the remote methods to be implemented by
 * URDS_proxy in USGF to interface with URDS created by Nanditha.
 *
 * @author Zhisheng Huang
 * @date January 2003
 * @version 1.0
 */
public interface IURDS_Proxy extends Remote
{
        public void searchConcreteComponents(AbstractComponent Component, String QID) throws
RemoteException;
        public void notifyClient(String msg,String QID) throws RemoteException;
        public void receiveQueryResult(Hashtable concreteComponentList,String resultType,String
QID)throws RemoteException;
}
```

MetaRepositoryHelper.java

```
import java.sql.*;
import java.util.*;

/**
 * Insert the type's description here.
 * Creation date: (05/18/2003 3:39:35 PM)
 * @author: Jayasree Gandhamaneni
 */
public class MetaRepositoryHelper
{
        private java.util.Hashtable resultTable;
```

```
        private QueryBean queryBean;

        public MetaRepositoryHelper() {
                super();
        }

        public MetaRepositoryHelper(QueryBean newQueryBean)
        {
                queryBean = newQueryBean;
        }

        public Hashtable getSearchResultTable(String componentTableName) throws java.lang.Exception
        {
                SQLHelper sqlHelper = new SQLHelper();
                String searchQuery = queryBean.getQuery(componentTableName);
        System.out.println("Query is "+searchQuery);

                if (searchQuery == null || searchQuery.equals(""))
                        throw new Exception("No Parameters Passed For Search");
        ResultSet resultSet = null;

        try
        {
                        resultSet = sqlHelper.executeQuery(searchQuery);
        }
        catch(Exception e)
        {
                        System.out.println("Error in executing query Headhunter Local
Metarepository"+e.getMessage());
                }

                // position to first record
                boolean moreRecords = resultSet.next();
                resultTable = new Hashtable();

                // If there are no records, display a message
                if (!moreRecords) {
                        sqlHelper.shutDown();
                        throw new Exception("No Records Matching Search Criteria");
                }
                else
                {
                        ConcreteComponent component = null;
        buildpersist buildcomponent= new buildpersist();

                        // get row data
                        do
                        {
                                String ID="";
                        try
                        {
                                        ID = resultSet.getString("id");//Get the host id
                        }
                        catch(Exception e) {
                                        System.out.println("Error in getting details of primary query"
+e.getMessage());
```

```
                                }
                                if (!resultTable.isEmpty() && resultTable.containsKey(ID)) {
                                        try {
                                                component = (ConcreteComponent)
resultTable.get(ID);
                                        }
                                        catch(Exception e)
                                        {
                                                System.out.println("Error in creation of Concrete
Component with resultTable not empty "+e.getMessage());
                                        }
                                }
                        else
                        {
                                        try
                                        {
                                        component = new ConcreteComponent();
                                                component= buildcomponent.build(component,
resultSet, componentTableName);
                                                resultTable.put(ID, component);
                                        } catch(Exception e){
                                                System.out.println("Error in building new Concrete
component :Metarepository"+e.getMessage());
                                        }
                                        }
                        } while (resultSet.next());

                } //end of else

                sqlHelper.shutDown();

                return resultTable;
        }
}
```

PrincipalAvailabilityChecker.java

```java
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;
import java.net.*;
import java.util.*;

/**
 * This class operates as a Thread which executes periodically
 * to check availability of active registries and headhunters
 * that are registered with the DomainSecurityManager.
 * Creation date: (06/15/2003 10:10:30 AM)
 * @author: Jayasree Gandhamaneni
 */

public class PrincipalAvailabilityChecker implements Runnable
{
        private long purgeTime = 0;
        private String dsmLocation = "";
        private IDomainSecurityManager dsm = null;
```

```
        /**
         * The PrincipalAvailabilityChecker Constructor.
         */
        public PrincipalAvailabilityChecker(
                long checkingTime,
                String dsmLocation) {

                try {

                        System.out.println("Thread started to check the availability of Headhunters and
ActiveRegistries");
                        System.setSecurityManager(new RMISecurityManager());
                        dsm = (IDomainSecurityManager) Naming.lookup(dsmLocation);
                        purgeTime = checkingTime;
                }catch (Exception e){
                        System.out.println("Exception in the constructor of
PrincipalAvailabilityChecker "+e);
                }
        }

        public void run()
        {
                Thread CurrentThread = Thread.currentThread();

                try {
                        while(true)
                        {
                                CurrentThread.sleep(purgeTime);
                                System.out.println("Checking availability of ARs and HHs");
                                long currentTime = (new java.util.Date()).getTime();

                                Hashtable arTable = new Hashtable();
                                Hashtable hhTable = new Hashtable();
                                arTable = dsm.getARTimestampTable();
                                hhTable = dsm.getHHTimestampTable();

                                Enumeration e = arTable.keys();

                                while (e.hasMoreElements()) {
                                        String key = (String) e.nextElement();
                                long registeredTime = ((Date) (arTable.get(key))).getTime();
                                        if((currentTime-registeredTime) > 2*purgeTime)
                                {

                                                arTable.remove(key);

                                }
                                }

                                Enumeration e1 = hhTable.keys();

                                while (e1.hasMoreElements()) {
                                        String key = (String) e1.nextElement();
                                long registeredTime = ((Date) (hhTable.get(key))).getTime();
                                        if((currentTime-registeredTime) > 2*purgeTime)
                                {
```

```
                                        hhTable.remove(key);

                            }
                            }

                        dsm.receiveUpdatedTables(arTable,hhTable);
                        System.out.println("Tables updated in DSM");

                }
            }catch(InterruptedException ie) {
                    System.out.println(ie.getMessage());
            }
            catch (Exception e)
            {
                    System.out.println(e);
            }
        }
}
```

QMAgent.java

```java
import de.ikv.grasshopper.communication.GrasshopperAddress;
import de.ikv.grasshopper.communication.ProxyGenerator;
import de.ikv.grasshopper.communication.ExternalCommService;
import de.ikv.grasshopper.type.Identifier;
import de.ikv.grasshopper.agency.AgentCreationFailedException;
import java.util.*;

public class QMAgent extends de.ikv.grasshopper.agent.MobileAgent
{

        private Hashtable resultTable=new Hashtable();
        private Hashtable agencyHHMapping=null;
        private String clientLocation=null;
        private GrasshopperAddress originatingAgencyAdd;
        private GrasshopperAddress targetAgencyAdd;
        private String ipAddress=null;
        private QueryBean queryBean=null;
        private GrasshopperAddress commObjectAddress;
        private IServerObject serverObjectProxy;
        private String qID = null;
        private int resultCounter = 0;
        private int randomResultResolver = 0;
        private boolean sendInterResults = false;


    public void init(Object[] args)
        {
                log("*******************");
        agencyHHMapping = (Hashtable)args[0];
                clientLocation = (String)args[1];
                originatingAgencyAdd = (GrasshopperAddress)args[2];
                targetAgencyAdd = originatingAgencyAdd;
                ipAddress = (String)targetAgencyAdd.toString();
        queryBean = (QueryBean)args[3];
                qID = (String)args[4];
```

```
                commObjectAddress = new
GrasshopperAddress("rmi://"+parseIPAddress(ipAddress)+":3000/ComSer");
                serverObjectProxy = (IServerObject)ProxyGenerator.newInstance(IServerObject.class,
                    commObjectAddress.generateAgentSystemId(),
                    commObjectAddress);

                int numberOfAgencies = agencyHHMapping.size();
                if(numberOfAgencies>5) {
                        randomResultResolver = (int)(Math.random()*(int)(numberOfAgencies/2))+1;
                        if( randomResultResolver > 1)
                        {
                                sendInterResults = true;
                        }
                }

        }

        /**
        * The name of the agent.
        */
        public String getName(){
        return "QMAgent";
        }


        /**
        * Callback from the agent system after moving.
        */
        public void afterMove(){
                try
                {
                        ipAddress=(String)targetAgencyAdd.toString();
                        commObjectAddress = new
GrasshopperAddress("rmi://"+parseIPAddress(ipAddress)+":3000/ComSer");
                        serverObjectProxy =
(IServerObject)ProxyGenerator.newInstance(IServerObject.class,
                                commObjectAddress.generateAgentSystemId(),
                                commObjectAddress);
                }
                catch(Exception e)
                {
                        log("error in afterMove method");
                        log(e.getMessage());
                }
        }

        /**
    * The lifecycle of this agent.
    */
        public void live(){
                log("QMAgent arrived at "+targetAgencyAdd);

        try {
                        ArrayList hhList = new ArrayList();
                        if(agencyHHMapping.size()>0 &&
```

```
agencyHHMapping.containsKey(targetAgencyAdd))
                                {
                                        hhList=(ArrayList)(agencyHHMapping.remove(targetAgencyAdd));
                                }

                        if(hhList.size()==0)
                        {
                                log("This system doesn't have any headhunters to visit");
                        }
                        else if(hhList.size()> 0)
                        {
                                resultCounter++;
                                log("Number of headhunters to be contacted are "+hhList.size());

                                Hashtable
tempCompData=serverObjectProxy.getCompDataFromHH(hhList,queryBean);
                                if(!tempCompData.isEmpty())
                                {
                                        Enumeration e1=tempCompData.keys();

                                        log("Components retrieved are as follows");

                                        while(e1.hasMoreElements())
                                        {
                                                String urlID=(String)e1.nextElement();
                                                ConcreteComponent component =
(ConcreteComponent)tempCompData.get(urlID);

                                                log("Component ID is  "+component.getID());
                                                resultTable.put(urlID,component);

                                        }
                                }
                                else
                                {
                                        log("No components found!!!");
                                }

                                if( sendInterResults==true &&
(resultCounter%randomResultResolver == 0))
                                {
                                        resultCounter = 0;

        serverObjectProxy.postCompDataToClient(clientLocation,resultTable,"inter",qID);
                                        resultTable.clear();
                                }
                        }

                        if(agencyHHMapping.size()> 0)
                        {
                                Enumeration e=agencyHHMapping.keys();
                                GrasshopperAddress tempAgencyAdd=null;

                                if(e.hasMoreElements())
                                {
                                        tempAgencyAdd=(GrasshopperAddress)e.nextElement();
                                }
```

```
                                originatingAgencyAdd=targetAgencyAdd;
                        targetAgencyAdd=(GrasshopperAddress)(tempAgencyAdd);
                        log("QMAgent moving to agency "+targetAgencyAdd);
                        move(targetAgencyAdd);
                        }
                        else
                        {
                                log("No more headhuters to visit");
                                log("Finally sending the component data to client "+clientLocation);

        serverObjectProxy.postCompDataToClient(clientLocation,resultTable,"final",qID);
                                try
                                {
                                        remove();
                                        log("QMAgent removed from the system");
                                }
                                catch(Exception e) {
                                        log("failed to remove Agent", e);
                                }

                        }
        } catch (Exception e) {
        log("Migration failed. Exception = ", e);
        serverObjectProxy.printMessage("QMAgent couldn't move.");
        }
        }

        private String parseIPAddress(String IPAddress)
        {
                int j=0;
                int start=0;
                int k=0;
                for (int i=0;i<IPAddress.length();i++)
                {
                        if(IPAddress.charAt(i)=='/')
                        {
                                k++;
                                if(k==2)
                                        start=i+1;
                        }
                else if(IPAddress.charAt(i)==':' && k>=2)
                {
                                j =i;
                i = IPAddress.length();
                }
                }
                return IPAddress.substring(start,j);
        }
}
```

QueryBean.java
import java.util.*; import java.io.*; /** * Insert the type's description here.

```
 * Creation date: (11/15/2001 1:15:26 PM)
 * @author: Nanditha Nayani, modified by Zhisheng in March 2003
 */

public class QueryBean implements Serializable
{
        private AbstractComponent component;
        private int numOffers = 0;
        private int numMetrics = 0;
        private int hopcount;
        private java.lang.String requestID;

        public QueryBean(AbstractComponent component)
        {
                this.component = component;
        }


        /**
        * Insert the method's description here.
        * Creation date: (11/15/2001 2:08:48 PM)
        * @return java.lang.String
        */
        public String getComponentNameQuery()
        {
                String searchQuery = " Componentname =" + "'"+ component.getComponentName() +
"'";
     return searchQuery;
        }

        public String getSystemNameQuery()
        {
                String systemNameQuery =
                "(" +
                " UPPER(systemName) = '" + component.getSystemName() + "' "+
                ")";

          return systemNameQuery;
        }

        public String getSubcaseQuery()
        {
                String subcaseQuery =" subcase = '" +component.getSubcase() + "' ";
                return subcaseQuery;
        }

        public String getIDQuery()
        {
                String idQuery =
                "(" +
                " UPPER(hostid) = " + component.getID() + "' "+
                ")";

          return idQuery;
        }
```

```
public String getVersionQuery()
{
        String VersionQuery =
        "(" +
        " UPPER(Version) = " + component.getVersion() + "' "+
        ")";

   return VersionQuery;
}

public String getAuthorQuery()
{
        String AuthorQuery =
        "(" +
        " UPPER(Author) = " + component.getAuthor() + "' "+
        ")";

   return AuthorQuery;
}

public String getDateQuery()
{
        String DateQuery =
        "(" +
        " UPPER(CreatingDate) = " + component.getDate() + "' "+
        ")";

   return DateQuery;
}

public String getValidityQuery()
{
        String ValidityQuery =
        "(" +
        " UPPER(Validity) = " + component.getValidity() + "' "+
        ")";

   return ValidityQuery;
}

public String getAtomicityQuery()
{
        String AtomicityQuery =
        "(" +
        " UPPER(Atomicity) = " + component.getAtomicity() + "' "+
        ")";

   return AtomicityQuery;
}

public String getRegistrationQuery()
{
        String RegistrationQuery =
        "(" +
        " UPPER(Registration) = " + component.getRegistration() + "' "+
        ")";
```

```
        return RegistrationQuery;
    }

    public String getModelQuery()
    {
            String ModelQuery =
            "(" +
            " UPPER(Model) = " + component.getModel() + "' "+
            ")";

        return ModelQuery;
    }

    public String getComplexityQuery()
    {
            String ComplexityQuery =
            "(" +
            " UPPER(Complexity) = " + component.getComplexity() + "' "+
            ")";

        return ComplexityQuery;
    }

    public String getSecurityQuery()
    {
            String SecurityQuery =
            "(" +
            " UPPER(Security) = " + component.getSecurity() + "' "+
            ")";

        return SecurityQuery;
    }

    public String getFaultToleranceQuery()
    {
            String FaultToleranceQuery =
            "(" +
            " UPPER(FaultTolerance) = " + component.getFaultTolerance() + "' "+
            ")";

        return FaultToleranceQuery;
    }

    public String getQoSLevelQuery()
    {
            String QoSLevelQuery =
            "(" +
            " UPPER(QoSLevel) = " + component.getQoSLevel() + "' "+
            ")";

        return QoSLevelQuery;
    }

    public String getCostQuery()
    {
```

```java
                String CostQuery =
                "(" +
                " UPPER(Cost) = " + component.getCost() + "' "+
                ")";

    return CostQuery;
}

public String getQualityLevelQuery()
{
                String QualityLevelQuery =
                "(" +
                " UPPER(QualityLevel) = " + component.getQualityLevel() + "' "+
                ")";

    return QualityLevelQuery;
}

/**
 * Insert the method's description here.
 * Creation date: (11/15/2001 5:08:43 PM)
 * @return java.lang.String
 */
public String getDomain()
{
                return component.getDomainName();
}

/**
 * Insert the method's description here.
 * Creation date: (11/15/2001 2:10:10 PM)
 * @return java.lang.String
 */
public String getMobilityQuery()
{
                String mobilityQuery =
                "(" +
                " UPPER(MOBILITY) LIKE '%" + component.getMobility().toUpperCase() + "%' "+
                ")";

    return mobilityQuery;
}

/**
 * Insert the method's description here.
 * Creation date: (11/15/2001 2:04:42 PM)
 * @param newNumMetrics int
 */
public void setNumMetrics(int newNumMetrics)
{
                numMetrics = newNumMetrics;
}

/**
 * Insert the method's description here.
 * Creation date: (11/15/2001 2:04:21 PM)
```

```
                * @param newNumOffers int
                */
                public void setNumOffers(int newNumOffers)
                {
                        numOffers = newNumOffers;
                }


                /**
                * Insert the method's description here.
                * Creation date: (11/15/2001 3:54:45 PM)
                * @return java.lang.String
                */
                public String[] tokeniseString(String keyWords)
                {
                        String[] stringTokens = null;

                        if (keyWords != null)
                        {
                                StringTokenizer strTok = new StringTokenizer(keyWords);
                                int numTokens = strTok.countTokens();
                                stringTokens = new String[numTokens];
                                int i = 0;
                                while (strTok.hasMoreTokens()) {
                                        stringTokens[i] = strTok.nextToken();
                                        i++;
                                }
                        }
                        return stringTokens;

                }

                /**
                * Insert the method's description here.
                * Creation date: (11/15/2001 2:19:41 PM)
                * @return java.lang.String
                */

                public String getQuery(String componentTableName)
                {
                        /* for experiments the following statement is commented and the new statement has
been used.
                        * while giving code in the appendix of project report, decomment the following
statement and
                        *remove the statement that is substituted for the following statement*/
                        String baseQuery = "SELECT * FROM " + componentTableName +
"UMMSpecification " +
                                                                " WHERE Domainname = 'Banking'";

                        //remove this later and decomment the above statement
                        //String baseQuery = "SELECT * FROM UMMSpecification WHERE Domainname =
'Banking'";


                        String bodyQuery = "";
```

```
                    if ((component.getComponentName() !=null) &&
(!component.getComponentName().equals("")))
                    {
                            bodyQuery = bodyQuery + " AND " + getComponentNameQuery();
                    }

                    if((component.getSubcase() != null) && (!component.getSubcase().equals("")))
                    {
                            bodyQuery = bodyQuery + " AND " + getSubcaseQuery();
                    }
                    String query = baseQuery + bodyQuery;

                    return query;

        }

        /**
         * Insert the method's description here.
         * Creation date: (3/16/02 8:39:02 PM)
         * @return int
         */
        public int getHopcount() {
                    return hopcount;
        }

        /**
         * Insert the method's description here.
         * Creation date: (3/16/02 8:39:34 PM)
         * @return java.lang.String
         */
        public java.lang.String getRequestID() {
                    return requestID;
        }

        /**
         * Insert the method's description here.
         * Creation date: (3/16/02 8:39:02 PM)
         * @param newHopcount int
         */
        public void setHopcount(int newHopcount) {
                    hopcount = newHopcount;
        }

        /**
         * Insert the method's description here.
         * Creation date: (3/16/02 8:39:34 PM)
         * @param newRequestID java.lang.String
         */
        public void setRequestID(java.lang.String newRequestID)
        {
                    requestID = newRequestID;
        }
}
```

QueryManager.java

```
import java.net.*;
import java.util.*;

import java.rmi.*;
import java.rmi.server.*;
import java.rmi.registry.*;

import java.security.*;
import java.io.*;
import java.lang.*;

import simpleCom.*;

import de.ikv.grasshopper.communication.GrasshopperAddress;
import de.ikv.grasshopper.communication.ExternalCommService;
import de.ikv.grasshopper.communication.ProxyGenerator;
import de.ikv.grasshopper.agency.IAgentSystem;
import de.ikv.grasshopper.agency.IRegionRegistration;
import de.ikv.grasshopper.agency.PlaceAlreadyExistsException;
import de.ikv.grasshopper.util.SearchFilter;
import de.ikv.grasshopper.type.AgentSystemInfo;
import de.ikv.grasshopper.type.AgentInfo;

/**
 * Insert the type's description here.
 * Creation date: (06/12/2003 05:36:30 PM)
 * @ author: Jayasree gandhamaneni
 */

public class QueryManager extends UnicastRemoteObject implements IQueryManager
{
        private IDomainSecurityManager dsm = null;
        private String qmLocation = null;
        private String regionRegistryAddress = null;

        /**
         * Remote method called by the URDS_Proxy to get components.
         */
        public void getSearchResultTable(QueryBean querybean, String clientLocation,String
qID)throws RemoteException
        {
                System.out.println("-------------------------");
                System.out.println("QM contacted by URDS_Proxy to propagate Search Query.");

                System.setSecurityManager(new RMISecurityManager());
                ArrayList hhList = dsm.getHHListForDomain(querybean.getDomain());

                System.out.println("QM obtained registered headhunter list from DSM for Domain " +
                        querybean.getDomain());
                if (hhList.size() == 0)
        {
                        try {
                                System.out.println("No Headhunters available for query propagation
in this domain ");
                                System.out.println("Notifying client about the unavailability of
headhunters");
```

```
                                    IURDS_Proxy urdsProxy = (IURDS_Proxy)
Naming.lookup(clientLocation);
                                    urdsProxy.notifyClient("No headhunters are available",qID);
                            } catch(Exception e){
                                    System.out.println("Exception in the getSearchResultTable() of
QueryManager\n"+e.getMessage());
                            }
        }
                    else if(hhList.size()>0)
                    {
                            ComponentSelectionAgent csAgent = new
ComponentSelectionAgent(hhList,querybean,clientLocation,regionRegistryAddress,qID);
                            Thread componentSelectionThread = new Thread(csAgent);
                            componentSelectionThread.start();

                    }

                    System.out.println("------------------------");

        }

        public static void main(String[] args) {

                    String qmLocation = "//"+args[0]+":"+ args[1] +"/QueryManager";
                    String dsmLocation = "//"+args[2]+":"+args[3]+"/DomainSecurityManager";
                    String regionRegistryAddress = "rmi://"+args[4]+":6020/MyRegion";

                    try
                    {
                            System.setSecurityManager(new RMISecurityManager());
                            Naming.rebind (qmLocation, new
QueryManager(dsmLocation,qmLocation,regionRegistryAddress));
                            System.out.println ("QueryManager is ready.");
                    }
                    catch (Exception e)
                    {
                            System.out.println ("QueryManager failed: " + e);
                    }
        }

        /**
         * The QueryManager Constructor.
         */
        public QueryManager(String dsmLoc, String qmLoc, String registryAddress) throws
RemoteException
        {
                    try
                    {
                            System.setSecurityManager(new RMISecurityManager());
                            dsm = (IDomainSecurityManager) Naming.lookup(dsmLoc);
                            qmLocation=qmLoc;
                            regionRegistryAddress = registryAddress;

                    } catch (Exception e) {
                            System.out.println(e.getMessage());
                    }
```

```
        } //end of constructor
}//end of QueryManager
```

ServerObject.java

```java
import java.util.*;
import java.rmi.*;
import java.rmi.server.UnicastRemoteObject;
import de.ikv.grasshopper.communication.GrasshopperAddress;

public class ServerObject implements IServerObject {

        private IDomainSecurityManager dsm = null;

        /**
         * The ServerObject Constructor.
         */
        public ServerObject(String dsmLocation)
        {
                try {
                        System.setSecurityManager(new RMISecurityManager());
                        dsm = (IDomainSecurityManager)Naming.lookup(dsmLocation);

                } catch (Exception e)
                {
                        System.out.println(e.getMessage());
                }
        }

        public void printMessage(String msg)
        {
        System.out.println("ServerObject receiving message: '" + msg + "'.");
        }

        /**
         * Remote method to return AR state to the HHAgent.
         */
        public int getARState(String arAddress)
        {
                int state=0;
                try {
                        IActiveRegistry myAR =(IActiveRegistry)Naming.lookup(arAddress);
                        state = myAR.getState();
                }catch(Exception e) {
                        System.out.println("Exception in getARState() of ServerObject: "
+e.getMessage());
                        e.printStackTrace();
                }
                return state;
        }

        /**
         * Remote method to get UniFrame Specification information
         * of components registered with a AR.
         */
```

```
        public Hashtable getCompDataFromAR(String arAddress, String headhunterLocation, String
mobileAgentUserName, String attributeType, String attributeValue)
        {
                Hashtable compData = new Hashtable();

                try {
                        IActiveRegistry myAR =(IActiveRegistry)Naming.lookup(arAddress);
                        compData=(Hashtable)(myAR.getComponentData(headhunterLocation,
mobileAgentUserName,attributeType,attributeValue));
                }
                catch(Exception e) {
                        System.out.println("Exception in getCompDataFromAR() of ServerObject: "
+e.getMessage());
                        e.printStackTrace();
                }

                return compData;
        }

        /**
         * Remote method to get UniFrame Specification information
         * of components available with a HH.
         */
        public Hashtable getCompDataFromHH(ArrayList hhList, QueryBean queryBean)
        {
                Hashtable compData = new Hashtable();

                try {

                        if(hhList.size()>0) {
                                if(hhList.size() == 1) {
                                        IHeadhunter myHH
=(IHeadhunter)Naming.lookup((String)hhList.get(0));
                                        compData=(Hashtable)(myHH.performSearch(queryBean));
                                        //System.out.println("Recieved  "+compData.size()+"
components from Headhunter "+hhList.get(0));
                                }
                                else if(hhList.size()>1) {
                                        Hashtable tempCompData = new Hashtable();

                                        while(!hhList.isEmpty()) {
                                                String hhAddress = (String)hhList.remove(0);
                                                //IHeadhunter myHH
=(IHeadhunter)Naming.lookup((String)hhList.remove(0));
                                                IHeadhunter myHH
=(IHeadhunter)Naming.lookup(hhAddress);
                                                System.out.println("ServerObject contacted  HH
"+hhAddress+" for registered services ");

        tempCompData=(Hashtable)(myHH.performSearch(queryBean));
                                                //System.out.println("Recieved
"+tempCompData.size()+" components from Headhunter ");

                                                Enumeration e=tempCompData.keys();
                                                //System.out.println("Components retrieved are as
follows");
```

```
                                                while(e.hasMoreElements()){
                                                        String urlID=(String)e.nextElement();
                                                        ConcreteComponent component =
(ConcreteComponent)tempCompData.get(urlID);
                                                        //System.out.println("Component
"+(String)component.getComponentName()+" available at "+urlID);
                                                        if( !compData.containsKey(urlID)) {
                                                                compData.put(urlID,component);
                                                        }
                                                }//end inner while
                                        }//end of outer while
                                }//end of else if
                }
                else {
                        System.out.println("No headhunters to contact on this machine");

                }

        }
        catch(Exception e) {
                System.out.println("Exception in the getCompDataFromHH() of ServerObject:
" +     e.getMessage());
                e.printStackTrace();
        }
        return compData;
}

/**
 * Remote method to return UniFrame Specification information
 * of components registered with a AR to the HH.
 */
public void postCompDataToHH(String hhLocation,Hashtable resultTable,String
resultType,long startTime,int noOfMsgs)
{
        try {
                IHeadhunter myHH =(IHeadhunter)Naming.lookup(hhLocation);
                myHH.populateMetaRepository(resultTable,resultType,startTime,noOfMsgs);

        } catch(Exception e){
                System.out.println("Exception in the method postCompDataToHH of
ServerObject : " +     e.getMessage());
                e.printStackTrace();
        }
}

/**
 * Remote method to return UniFrame Specification information
 * of components available with a HH to a client.
 */

public void postCompDataToClient(String clientLocation,Hashtable resultTable,String
resultType, String qID)
{
        try {
                //instead of contacting QM, contact urds_proxy
```

```
                        IURDS_Proxy urdsProxy = (IURDS_Proxy)Naming.lookup(clientLocation);
                        urdsProxy.receiveQueryResult(resultTable,resultType,qID);

              } catch(Exception e) {
                        System.out.println("Exception in the method
postCompDataToClient of ServerObject : " +e.getMessage());
                        e.printStackTrace();
              }
   }
}
```

SQLHelper.java

```
import java.util.*;
import java.sql.*;

/**
 * This is the class which serves as a connection to the
 * oracle database. It establises the database connection
 * and executes queries which either select/update the
 * tables of the database as well as execute stored
 * procedures.
 * Creation date: (9/14/2001 12:57:07 PM)
 * @author: Nanditha Nayani
 */
public class SQLHelper {
        private java.sql.Connection dbconn = null;
        private java.sql.Statement statement = null;

/**
 * The SQLHelper Constructor.
 */

public SQLHelper() throws java.lang.Exception {

        //-------------------------------------------------------------------
        // Get Connection to database.
        // The URL specifying the database to which
        // this program connects using JDBC
        //-------------------------------------------------------------------

        String url = "jdbc:oracle:thin:@phoenix.cs.iupui.edu:1521:cs9iorcl";

        String username = "jgandham";
        String password = "mobileurds";

        // Load the driver to allow connection to the database
        try {
                Class.forName("oracle.jdbc.driver.OracleDriver");
                dbconn = DriverManager.getConnection(url, username, password);
                statement = dbconn.createStatement();
        } catch (ClassNotFoundException cnfex) {
                System.err.println("Failed to load driver.");
                cnfex.printStackTrace();
                System.exit(1); // terminate program
```

```java
		} catch (SQLException sqlex) {

				System.err.println("\n Unable to connect to Oracle Server");
				sqlex.printStackTrace();
				System.exit(1); // terminate program
		}

}

/**
 * Commit Transaction.
 */
public final void commitTransaction() throws java.lang.Exception {
		try {
				dbconn.commit();
		} catch (SQLException sqle) {
				throw new Exception(
						"\n SQL Exception during commitTransaction with message :" +
sqle.getMessage());
		}
}

/**
 * Execute a query.
 */
public ResultSet executeQuery(String query) throws java.lang.Exception {

		ResultSet resultSet = null;

		try {
				resultSet = statement.executeQuery(query);
		} catch (SQLException sqle) {
				throw new Exception(
						"\n SQL Exception during executeQuery with message:" + sqle.getMessage());
		}

		return resultSet;

}

/**
 * Return DB connection.
 */
public java.sql.Connection getDbconn() {
		return dbconn;
}

/**
 * Return statement.
 */
public java.sql.Statement getStatement() {
		return statement;
}

/**
```

```java
 * Turn off Auto Commit before initiating transaction.
 */
public final void initiateTransaction() throws java.lang.Exception {
        try {
                dbconn.setAutoCommit(false);
        } catch (SQLException sqle) {
                throw new Exception(
                        "\n SQL Exception during initiateTransaction with message :"
                                + sqle.getMessage());
        }
}

/**
 * Perform Rollback.
 */
public void rollbackTransaction() throws java.lang.Exception {
        try {
                dbconn.rollback();
        } catch (SQLException sqle) {
                throw new Exception(
                        "\n SQL Exception during rollbackTransaction with message :"
                                + sqle.getMessage());
        }
}

/**
 * Set DB connection.
 */
public void setDbconn(java.sql.Connection newDbconn) {
        dbconn = newDbconn;
}

/**
 * Set statement.
 */
public void setStatement(java.sql.Statement newStatement) {
        statement = newStatement;
}

/**
 * Close connection.
 */
public void shutDown() throws java.lang.Exception {

        try {

                if (statement != null){
                        //System.out.println("inside of the shutdown method  statement!=null");
                        statement.close();
                }
                if (dbconn != null)
                {
                        //System.out.println("inside of the shutdown method, dbconn!=null");
                        dbconn.close();
                }
```

```
            } catch (Exception e) {
                    throw new Exception(e.getMessage());
            }


}

/**
 * Update DB Table.
 */
public void updateTable(String updateString) throws java.lang.Exception {

        try {
                //System.out.println("in updateTable");
                dbconn.setAutoCommit(false);
                //System.out.println(updateString);
                statement.executeUpdate(updateString.trim());
                //System.out.println("statement succeeded");
                dbconn.commit();
        } catch (SQLException sqle) {
                throw new Exception(
                        "\n SQL Exception from function updateTable with message:" +
sqle.getMessage());
        }
}}
```

SystemQoS.java

```
import java.util.*;
import java.io.*;

/**
 * This class stores system QoS in a Hashtable. The keys of the Hashtable are
 * QoS parameters.
 *
 * @author Zhisheng Huang
 * @date January 2003
 * @version 1.0
 */
public class SystemQoS implements Serializable
{
   private String systemName;
   private Hashtable systemQoS;

   /**
    * Constructor.
    */
   public SystemQoS(String systemName)
   {
     this.systemName = systemName;
     systemQoS = new Hashtable();
   }

   /**
    * This method add a pair of QoS parameter and its value. The first argument is
    * the system name, which is for error checking.
    */
```

```java
public void addSystemQoS(String systemName, String QoSParameter, String value)
{
    if(systemName.equals(this.systemName))
    {
        systemQoS.put(QoSParameter, value);
    }
}

/**
 * This method gets the QoS value for a QoS parameter. The first argument is
 * the system name, which is for error checking.
 */
public String getSystemQoS(String systemName, String QoSParameter)
{
    if(systemName.equals(this.systemName))
    {
        return (String)systemQoS.get(QoSParameter);
    }
    else
    {
        return null;
    }
}
}
```

UniFrameIntrospector.java

```java
import javax.servlet.http.*;
import java.lang.reflect.*;
import java.beans.*;
import java.util.*;

/**
 * Creation date: (6/11/2001 9:18:51 AM)
 * @author: Nanditha Nayani
 */
public class UniFrameIntrospector {
public UniFrameIntrospector() {
        super();
}


public static Object getProperty(Object bean, String propertyName)
        throws UniFrameIntrospectorException {

        Object property = null;
        Method method = null;
        Object[] args = null;

        try {

                BeanInfo info = Introspector.getBeanInfo(bean.getClass());
                PropertyDescriptor[] pds = info.getPropertyDescriptors();

                for (int i = 0; pds != null && i < pds.length; i++) {
                        if (pds[i].getName().equals(propertyName)) {
```

```
                                    method = pds[i].getReadMethod();
                                    break;
                            }
                    }
            } catch (IntrospectionException e) {
                    throw new UniFrameIntrospectorException(
                            "Error analyzing the bean class: " + e.getMessage());
            }

            if (method == null)
                    throw new UniFrameIntrospectorException(
                            "Property " + propertyName + " not found");

            try {

                    property = method.invoke(bean, args);
             System.out.println("Active Registry obtained UMM Spec URL by Introspection : " + property);

            } catch (Exception e) {
                    throw new UniFrameIntrospectorException(
                            e.getClass().getName()
                                    + ": "
                                    + "Failed to get property "
                                    + propertyName
                                    + ", message: "
                                    + e.getMessage());
            }

            return property;
    }
}
```

UniFrameIntrospectorException.java

```
/**
 * Creation date: (10/5/2001 12:55:32 PM)
 * @author: Nanditha Nayani
 */
public class UniFrameIntrospectorException extends Exception {
public UniFrameIntrospectorException() {
        super();
}
public UniFrameIntrospectorException(String s) {
        super(s);
}
}
```

UniFrameSpecificationParser.java

```
import java.io.IOException;
import org.w3c.dom.Document;
import org.w3c.dom.DocumentType;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.apache.xerces.parsers.DOMParser;
```

```
import java.util.*;

/*
 *
 */
public class UniFrameSpecificationParser
{
   private ConcreteComponent component;
   private Vector functionVector = new Vector();
   private boolean populatedFlag = false;
   private Vector syntaxVector = new Vector();

   public UniFrameSpecificationParser(String url)
   {

      // Instantiate the vendor's DOM parser implementation
      DOMParser parser = new DOMParser();

      try
      {
         parser.parse(url);
         Document doc = parser.getDocument();

         // Parse the document from the DOM tree.
         NodeList children = doc.getChildNodes();

         if (children != null)
         {
            for (int i = 0; i < children.getLength(); i++)
            {
               if(children.item(i).getNodeName().equalsIgnoreCase("UMM_ConcreteComponent"))
               {
                  parseUMMSpecification(children.item(i));
                  break; //allow each file contain only one architecture model
               }
            }
         }
      }
      catch (IOException e)
      {
         System.out.println("Error reading URL: " + e.getMessage());
      }
      catch (Exception ex)
      {
         System.out.println("Error in parsing: " + ex.getMessage());
      }
   }
   public ConcreteComponent getConcreteComponent()
   {
      return component;
   }

   private void parseUMMSpecification(Node node)
   {
      NodeList children = node.getChildNodes();
```

```
     if(children == null)
       return;

    component = new ConcreteComponent();

    for (int i = 0; i < children.getLength(); i++)
    {
       if(children.item(i).getNodeName().equalsIgnoreCase("componentname"))
       {
          component.setComponentName(children.item(i).getFirstChild().getNodeValue().trim());
       }
       else if(children.item(i).getNodeName().equalsIgnoreCase("componentSubcase"))
       {
          component.setSubcase(children.item(i).getFirstChild().getNodeValue().trim());
       }
       else if(children.item(i).getNodeName().equalsIgnoreCase("domainname"))
       {
          component.setDomainName(children.item(i).getFirstChild().getNodeValue().trim());
       }
       else if(children.item(i).getNodeName().equalsIgnoreCase("systemname"))
       {
          component.setSystemName(children.item(i).getFirstChild().getNodeValue().trim());
       }
       else if(children.item(i).getNodeName().equalsIgnoreCase("description"))
       {
          component.setDescription(children.item(i).getFirstChild().getNodeValue().trim());
       }
       else if(children.item(i).getNodeName().equalsIgnoreCase("ComputationalAttributes"))
       {
          NodeList children_computationalAttributes = children.item(i).getChildNodes();

          for(int j = 0; j < children_computationalAttributes.getLength(); j++)
          {
             if(children_computationalAttributes.item(j).getNodeName().equals("InherentAttributes"))
             {
               NodeList children_inherentAttributes =
children_computationalAttributes.item(j).getChildNodes();

               for(int k = 0; k < children_inherentAttributes.getLength(); k++)
               {
                  if(children_inherentAttributes.item(k).getNodeName().equalsIgnoreCase("id"))
                  {

component.setID(children_inherentAttributes.item(k).getFirstChild().getNodeValue().trim());
                  }
                  else
if(children_inherentAttributes.item(k).getNodeName().equalsIgnoreCase("version"))
                  {

component.setVersion(children_inherentAttributes.item(k).getFirstChild().getNodeValue().trim());
                  }
                  else
if(children_inherentAttributes.item(k).getNodeName().equalsIgnoreCase("author"))
                  {

component.setAuthor(children_inherentAttributes.item(k).getFirstChild().getNodeValue().trim());
```

```
                        }
                    else if(children_inherentAttributes.item(k).getNodeName().equalsIgnoreCase("date"))
                        {

component.setDate(children_inherentAttributes.item(k).getFirstChild().getNodeValue().trim());
                        }
                    else
if(children_inherentAttributes.item(k).getNodeName().equalsIgnoreCase("validity"))
                        {

component.setValidity(children_inherentAttributes.item(k).getFirstChild().getNodeValue().trim());
                        }
                    else
if(children_inherentAttributes.item(k).getNodeName().equalsIgnoreCase("atomicity"))
                        {

component.setAtomicity(children_inherentAttributes.item(k).getFirstChild().getNodeValue().trim());
                        }
                    else
if(children_inherentAttributes.item(k).getNodeName().equalsIgnoreCase("registration"))
                        {

component.setRegistration(children_inherentAttributes.item(k).getFirstChild().getNodeValue().trim());
                        }
                    else if(children_inherentAttributes.item(k).getNodeName().equalsIgnoreCase("model"))
                        {

component.setModel(children_inherentAttributes.item(k).getFirstChild().getNodeValue().trim());
                        }
                    }
                }
            else
if(children_computationalAttributes.item(j).getNodeName().equals("FunctionalAttributes"))
                {
                  NodeList children_functionalAttributes =
children_computationalAttributes.item(j).getChildNodes();

                  for(int k = 0; k < children_functionalAttributes.getLength(); k++)
                    {
                      if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("purpose"))
                        {

component.setPurpose(children_functionalAttributes.item(k).getFirstChild().getNodeValue().trim());
                        }
                    else
if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("algorithms"))
                        {
                          NodeList children_algorithms =
children_functionalAttributes.item(k).getChildNodes();
                          ArrayList algorithms = new ArrayList();

                          for(int l = 0; l < children_algorithms.getLength(); l++)
                            {
                              if(children_algorithms.item(l).getNodeName().equalsIgnoreCase("algorithm"))
                                {
                                  String value =
```

```
children_algorithms.item(l).getFirstChild().getNodeValue().trim();
                            if(!value.equals(""))
                            {
                                algorithms.add(value);
                            }
                        }
                    }

                    if(algorithms.size() != 0)
                    {
                        component.setAlgorithms((String[])algorithms.toArray(new String[1]));
                    }
                }
                else
if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("complexity"))
                {

component.setComplexity(children_functionalAttributes.item(k).getFirstChild().getNodeValue().trim());
                }
                else
if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("SyntacticContract"))
                {
                    NodeList children_syntax = children_functionalAttributes.item(k).getChildNodes();

                    for(int l =  0; l < children_syntax.getLength(); l++)
                    {
if(children_syntax.item(l).getNodeName().equalsIgnoreCase("ProvidedInterfaces"))
                        {
                            NodeList children_provided = children_syntax.item(l).getChildNodes();
                            ArrayList providedInterfaces = new ArrayList();

                            for(int m = 0; m < children_provided.getLength(); m++)
                            {
                                if(children_provided.item(m).getNodeName().equalsIgnoreCase("Interface"))
                                {
                                    String value =
children_provided.item(m).getFirstChild().getNodeValue().trim();
                                    if(!value.equals(""))
                                        providedInterfaces.add(value);
                                }
                            }

                            if(providedInterfaces.size() != 0)
                            {
                                component.setProvidedInterfaces((String[])providedInterfaces.toArray(new
String[1]));
                            }
                        }
                        else
if(children_syntax.item(l).getNodeName().equalsIgnoreCase("RequiredInterfaces"))
                        {
                            NodeList children_required = children_syntax.item(l).getChildNodes();
                            ArrayList requiredInterfaces = new ArrayList();

                            for(int m = 0; m < children_required.getLength(); m++)
```

```
                                    {
                                      if(children_required.item(m).getNodeName().equalsIgnoreCase("Interface"))
                                      {
                                          String value =
children_required.item(m).getFirstChild().getNodeValue().trim();
                                          if(!value.equals(""))
                                             requiredInterfaces.add(value);
                                      }
                                    }

                                    if(requiredInterfaces.size() != 0)
                                    {
                                        component.setRequiredInterfaces((String[])requiredInterfaces.toArray(new
String[1]));
                                    }
                                }
                            }
                        }
                        else
if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("technologies"))
                        {
                            NodeList children_technologies =
children_functionalAttributes.item(k).getChildNodes();
                            ArrayList technologies = new ArrayList();

                            for(int l = 0; l < children_technologies.getLength(); l++)
                            {
                                if(children_technologies.item(l).getNodeName().equalsIgnoreCase("technology"))
                                {
                                    String value =
children_technologies.item(l).getFirstChild().getNodeValue().trim();
                                    if(!value.equals(""))
                                    {
                                        technologies.add(value);
                                    }
                                }
                            }

                            if(technologies.size() != 0)
                            {
                                component.setTechnologies((String[])technologies.toArray(new String[1]));
                            }
                        }
                        else
if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("expectedResources"))
                        {
                            NodeList children_resources =
children_functionalAttributes.item(k).getChildNodes();
                            ArrayList resources = new ArrayList();

                            for(int l = 0; l < children_resources.getLength(); l++)
                            {
                                if(children_resources.item(l).getNodeName().equalsIgnoreCase("resource"))
                                {
                                    String value = children_resources.item(l).getFirstChild().getNodeValue().trim();
                                    if(!value.equals(""))
```

```
                    {
                       resources.add(value);
                    }
                 }
              }

              if(resources.size() != 0)
              {
                 component.setExpectedResources((String[])resources.toArray(new String[1]));
              }
           }
           else
if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("designPatterns"))
           {
              NodeList children_patterns = children_functionalAttributes.item(k).getChildNodes();
              ArrayList patterns = new ArrayList();

              for(int l = 0; l < children_patterns.getLength(); l++)
              {
                 if(children_patterns.item(l).getNodeName().equalsIgnoreCase("pattern"))
                 {
                    String value = children_patterns.item(l).getFirstChild().getNodeValue().trim();
                    if(!value.equals(""))
                    {
                       patterns.add(value);
                    }
                 }
              }

              if(patterns.size() != 0)
              {
                 component.setDesignPatterns((String[])patterns.toArray(new String[1]));
              }
           }
           else
if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("knownUsage"))
           {
              NodeList children_usages = children_functionalAttributes.item(k).getChildNodes();
              ArrayList usages = new ArrayList();

              for(int l = 0; l < children_usages.getLength(); l++)
              {
                 if(children_usages.item(l).getNodeName().equalsIgnoreCase("usage"))
                 {
                    String value = children_usages.item(l).getFirstChild().getNodeValue().trim();
                    if(!value.equals(""))
                    {
                       usages.add(value);
                    }
                 }
              }

              if(usages.size() != 0)
              {
                 component.setKnownUsages((String[])usages.toArray(new String[1]));
              }
```

```
                    }
                    else
if(children_functionalAttributes.item(k).getNodeName().equalsIgnoreCase("aliases"))
                    {
                        NodeList children_aliases = children_functionalAttributes.item(k).getChildNodes();
                        ArrayList aliases = new ArrayList();

                        for(int l = 0; l < children_aliases.getLength(); l++)
                        {
                            if(children_aliases.item(l).getNodeName().equalsIgnoreCase("alias"))
                            {
                                String value = children_aliases.item(l).getFirstChild().getNodeValue().trim();
                                if(!value.equals(""))
                                {
                                    aliases.add(value);
                                }
                            }
                        }

                        if(aliases.size() != 0)
                        {
                            component.setAliases((String[])aliases.toArray(new String[1]));
                        }
                    }
                }
            }
        }
    }
    else if(children.item(i).getNodeName().equalsIgnoreCase("CooperationAttributes"))
    {
        NodeList children_cooperativeAttributes = children.item(i).getChildNodes();

        for(int j = 0; j < children_cooperativeAttributes.getLength(); j++)
        {
if(children_cooperativeAttributes.item(j).getNodeName().equalsIgnoreCase("preprocessingCollaborators"
))
            {
                NodeList children_preprocessingCollaborators =
children_cooperativeAttributes.item(j).getChildNodes();
                ArrayList collaborators = new ArrayList();

                for(int k = 0; k < children_preprocessingCollaborators.getLength(); k++)
                {
if(children_preprocessingCollaborators.item(k).getNodeName().equalsIgnoreCase("Collaborator"))
                    {
                        String value =
children_preprocessingCollaborators.item(k).getFirstChild().getNodeValue().trim();
                        if(!value.equals(""))
                            collaborators.add(value);
                    }
                }

                if(collaborators.size() != 0)
                {
```

```
                    component.setPreProcessingCollaborators((String[])collaborators.toArray(new
String[1]));
                }
            }
            else
if(children_cooperativeAttributes.item(j).getNodeName().equalsIgnoreCase("postprocessingCollaborators
"))
            {
                NodeList children_postprocessingCollaborators =
children_cooperativeAttributes.item(j).getChildNodes();
                ArrayList collaborators = new ArrayList();

                for(int k = 0; k < children_postprocessingCollaborators.getLength(); k++)
                {

if(children_postprocessingCollaborators.item(k).getNodeName().equalsIgnoreCase("Collaborator"))
                    {
                        String value =
children_postprocessingCollaborators.item(k).getFirstChild().getNodeValue().trim();
                        collaborators.add(value);
                    }
                }

                if(collaborators.size() != 0)
                {
                    component.setPostProcessingCollaborators((String[])collaborators.toArray(new
String[1]));
                }
            }
        }
    }
    else if(children.item(i).getNodeName().equalsIgnoreCase("AuxiliaryAttributes"))
    {
        NodeList children_auxiliaryAttributes = children.item(i).getChildNodes();

        for(int j = 0; j < children_auxiliaryAttributes.getLength(); j++)
        {
            if(children_auxiliaryAttributes.item(j).getNodeName().equalsIgnoreCase("Mobility"))
            {

component.setMobility(children_auxiliaryAttributes.item(j).getFirstChild().getNodeValue().trim());
            }
            else if(children_auxiliaryAttributes.item(j).getNodeName().equalsIgnoreCase("Security"))
            {

component.setSecurity(children_auxiliaryAttributes.item(j).getFirstChild().getNodeValue().trim());
            }
            else
if(children_auxiliaryAttributes.item(j).getNodeName().equalsIgnoreCase("FaultTolerance"))
            {

component.setFaultTolerance(children_auxiliaryAttributes.item(j).getFirstChild().getNodeValue().trim());
            }
        }
    }
    else if(children.item(i).getNodeName().equalsIgnoreCase("QoS"))
```

```java
        {
            NodeList children_qos = children.item(i).getChildNodes();

            for(int j = 0; j < children_qos.getLength(); j++)
            {
                if(children_qos.item(j).getNodeName().equalsIgnoreCase("QoSMetrics"))
                {
                    NodeList children_qosMetrics = children_qos.item(j).getChildNodes();
                    ComponentQoS componentQoS = new ComponentQoS(component.getSystemName(),
component.getComponentName());

                    for(int k = 0; k < children_qosMetrics.getLength(); k++)
                    {
                        String parameterName = null;
                        String functionName = null;
                        String value = null;

                        if(children_qosMetrics.item(k).getNodeName().equalsIgnoreCase("metric"))
                        {
                            NodeList children_metric = children_qosMetrics.item(k).getChildNodes();

                            for(int l = 0; l < children_metric.getLength(); l++)
                            {
                                if(children_metric.item(l).getNodeName().equalsIgnoreCase("ParameterName"))
                                {
                                    parameterName =
children_metric.item(l).getFirstChild().getNodeValue().trim();
                                }
                                else
if(children_metric.item(l).getNodeName().equalsIgnoreCase("FunctionName"))
                                {
                                    functionName = children_metric.item(l).getFirstChild().getNodeValue().trim();
                                }
                                else if(children_metric.item(l).getNodeName().equalsIgnoreCase("Value"))
                                {
                                    value = children_metric.item(l).getFirstChild().getNodeValue().trim();
                                }
                            }
                        }

                        if(parameterName != null && functionName != null && value != null &&
!parameterName.equals("") && !functionName.equals("") && !value.equals(""))
                        {
                            FunctionQoS functionQoS =
(FunctionQoS)componentQoS.getFunctionQoS(component.getComponentName(), functionName);
                            if(functionQoS == null)
                            {
                                functionQoS = new FunctionQoS(component.getComponentName(),
functionName);
                            }
                            functionQoS.addFunctionQoS(component.getComponentName(), functionName,
parameterName, value);
                            componentQoS.addFunctionQoS(functionQoS);
                        }
                    }
```

```
                    component.setComponentQoS(componentQoS);
            }
            else if(children_qos.item(j).getNodeName().equalsIgnoreCase("QoSLevel"))
            {
                component.setQoSLevel(children_qos.item(j).getFirstChild().getNodeValue().trim());
            }
            else if(children_qos.item(j).getNodeName().equalsIgnoreCase("cost"))
            {
                component.setCost(children_qos.item(j).getFirstChild().getNodeValue().trim());
            }
            else if(children_qos.item(j).getNodeName().equalsIgnoreCase("QualityLevel"))
            {
                component.setQualityLevel(children_qos.item(j).getFirstChild().getNodeValue().trim());
            }
        }
      }
    }
    //put code here to check validity to bulletproof the system
    //or the code can be inserted in the appropriate places above.
  }
}
```

URDS_Proxy.java

```java
import java.rmi.*;
import java.rmi.server.*;
import java.net.*;
import java.util.*;


/**
 * This class serves as the client that sends queries to the QueryManager.
 * @author Jayasree Gandhamaneni
 * @date March 2004
 */
public class URDS_Proxy extends UnicastRemoteObject implements IURDS_Proxy
{
        IQueryManager queryManager = null;
        String proxyLocation;
        Hashtable queryTable=new Hashtable();
        long timer =0;
        int queryCounter = 0;
        public URDS_Proxy(String qmAddress,String qmPort, String url) throws RemoteException
        {
        try
        {
                        queryManager =
(IQueryManager)Naming.lookup("//"+qmAddress+":"+qmPort+"/QueryManager");
                        proxyLocation = url;
                }
        catch(RemoteException e)
        {
        System.err.println(e);
        }
        catch(NotBoundException e)
        {
```

```
            System.err.println(e);
            }
        catch(MalformedURLException e)
        {
        System.err.println(e);
        }
        }

        public void searchConcreteComponents(AbstractComponent Component,String qID) throws
RemoteException
        {
        if(queryManager == null)
        {
        System.out.println("QueryManager is not ready");
                        System.exit(0);
        }

        QueryBean queryBean = new QueryBean(Component);
                queryManager.getSearchResultTable(queryBean,proxyLocation,qID);
                queryTable.put(qID,new Long(System.currentTimeMillis()));

        }

        public void notifyClient(String msg,String qID) {
                System.out.println(msg+" for query "+ qID);
                long endTime = (new java.util.Date()).getTime();
                long startTime = ((Long)queryTable.get(qID)).longValue();
                System.out.println("Total time taken for query "+qID+" is "+ (endTime-startTime));

                timer+=endTime-startTime;
                queryCounter ++;
                if(queryCounter == queryTable.size())
                {
                        long avgTime = timer/queryCounter;
                        System.out.println("Avg. time taken to process "+queryTable.size()+" queries
is "+avgTime);
                        System.exit(0);
                }
        }


        public void receiveQueryResult(Hashtable concreteComponentList,String resultType,String qID)
{
                ArrayList resultList = new ArrayList();
                try {
                        long endTime = (new java.util.Date()).getTime();
                        long startTime = ((Long)queryTable.get(qID)).longValue();
                        System.out.println("---------Results for "+qID+"----------");

                        System.out.println("Total time taken for query "+qID+" is "+ (endTime-
startTime));
                        timer+=endTime-startTime;
                        queryCounter ++;

                        Enumeration e = concreteComponentList.elements();
                        if(e.hasMoreElements())
```

```
                                    {
                                            System.out.println("Number of components found for "+qID+" are
"+concreteComponentList.size());

                                            int counter=0;
                                            while(e.hasMoreElements()){
                                                    counter++;
                                                    ConcreteComponent component
=(ConcreteComponent)e.nextElement();

        System.out.println(counter+"."+component.getComponentName());
                                            }
                                    }
                                    else
                                    {
                                            System.out.println("No components found that match the
query"+qID);
                                    }

                                    if(queryCounter == queryTable.size())
                                    {
                                            long avgTime = timer/queryCounter;
                                            System.out.println("Avg. time taken to process "+queryTable.size()+"
queries is "+avgTime);

                                            System.exit(0);

                                    }
                                    System.out.println("------------------");
                    }
                    catch(Exception e) {
        System.out.println(e.getMessage());
                    }
                    }

    public AbstractComponent Query1() {
                    AbstractComponent component = new AbstractComponent();
            component.setDomainName("Banking");
                    return component;
    }

    public AbstractComponent Query2() {
                    AbstractComponent component = new AbstractComponent();
            component.setDomainName("Banking");
            component.setComponentName("AccountDatabase");
                    return component;
    }

    public AbstractComponent Query3() {
                    AbstractComponent component = new AbstractComponent();
            component.setDomainName("Banking");
            component.setComponentName("ATM");
                    return component;
    }

    public AbstractComponent Query4() {
                    AbstractComponent component = new AbstractComponent();
            component.setDomainName("Banking");
```

```java
        component.setComponentName("CashierValidationServer");
            return component;
    }

  public AbstractComponent Query5() {
            AbstractComponent component = new AbstractComponent();
component.setDomainName("Banking");
component.setComponentName("CustomerValidationServer");
            return component;
    }

  public AbstractComponent Query6() {
            AbstractComponent component = new AbstractComponent();
component.setDomainName("Banking");
component.setComponentName("DeluxeTransactionServer");
            return component;
    }

  public AbstractComponent Query7() {
            AbstractComponent component = new AbstractComponent();
component.setDomainName("Banking");
component.setComponentName("EconomicTransactionServer");
            return component;
    }

  public AbstractComponent Query8() {
            AbstractComponent component = new AbstractComponent();
component.setDomainName("Banking");
component.setComponentName("TransactionServerManager");
            return component;
    }


  public AbstractComponent Query9() {
            AbstractComponent component = new AbstractComponent();
component.setDomainName("Banking");
component.setComponentName("CashierTerminal");
            return component;
    }

  public AbstractComponent Query10() {
            AbstractComponent component = new AbstractComponent();
component.setDomainName("Banking");
            return component;
    }

  public static void main(String[] args) {

            String url = "//" + args[0]+":"+args[1]+"/URDS_Proxy";

  try
  {
  URDS_Proxy urds_Proxy= new URDS_Proxy(args[2],args[3],url);
                    int numberOfIterations = Integer.parseInt(args[4]);
  Naming.rebind(url, urds_Proxy);
```

```
                    ArrayList queryList = new ArrayList();
                    queryList.add(urds_Proxy.Query1());
                    queryList.add(urds_Proxy.Query2());
                    queryList.add(urds_Proxy.Query3());
                    queryList.add(urds_Proxy.Query4());
                    queryList.add(urds_Proxy.Query5());
                    queryList.add(urds_Proxy.Query6());
                    queryList.add(urds_Proxy.Query7());
                    queryList.add(urds_Proxy.Query8());
                    queryList.add(urds_Proxy.Query9());
                    queryList.add(urds_Proxy.Query10());

                    for(int i=0;i<numberOfIterations;i++)
                    {
                            for(int j=0;j<queryList.size();j++)
                            {
    urds_Proxy.searchConcreteComponents((AbstractComponent)queryList.get(j),("Q"+i+j));
                            }
                    }
        } catch(RemoteException e) {
                System.err.println(e.getMessage());
        }
        catch(MalformedURLException e)
        {
        System.err.println(e.getMessage());
        }
        catch(Exception e)
        {
        System.err.println(e);
        System.exit(1);
        }
        }
}
```

APPENDIX H: Commands To Run the System

* Order to start the entities of the MURDS System

    * RegionRegistry
    * Agency
    * DomainSecurityManager
    * ExternalCommunicationServer
    * One or more ActiveRegistries
    * Components to be registered with each ActiveRegistry
    * One or more Headhunters
    * One or more QueryManagers
    * URDS_Proxy

* Environment parameters

classpath

.;C:\murds\CompRep\classes;C:\Grasshopper2.2.4\examples\classes\examples;
C:\Grasshopper2.2.4\lib\gh.jar;C:\Grasshopper2.2.4\lib\jasper.jar;C:\Grasshopper2.2.4\lib\jaxp.jar;
C:\Grasshopper2.2.4\lib\jndi.jar;C:\Grasshopper2.2.4\lib\ldap.jar;C:\Grasshopper2.2.4\lib\parser.jar;
C:\Grasshopper2.2.4\lib\servlet.jar;C:\Grasshopper2.2.4\lib\tomcat.jar;C:\murds\classes111.zip;
C:\murds\xerces.jar;C:\murds;

path

C:\Grasshopper2.2.4\bin;C:\murds\CompRep\classes;

Install Grasshopper software on a system and set the environment parameters as specified above.
Then poen a MS-DOS prompt and type Grasshopper at command prompt. The grasshopper system
opens a wizard and guides to start either a Region Registry or an Agency. Choose one of them.

* RegionRegistry

For a Region Registry, an example of the properties that needs to be specified in the Edit Profile
window of the wizard is mentioned below...
Default Address   192.168.0.100
Default Port   6020
Default Protocol   rmi
Graphical UI   True
Name   MyRegion
Textual UI   True
Trace Errors   True
Trace Warnings   True
Note: Always set the Default Port to 6020 and Name to MyRegion as these are hard coded in the
Headhunter and QueryMAnager classes. Otherwise, a Headhunter or a QueryManager trows an
error.

- Agency

For an Agency, an example of the properties that needs to be specified in the Edit Profile window of the wizard is mentioned below...

Default Address   192.168.0.100
Default Port   6000
Default Protocol   rmi
Graphical UI   True
Name   MyAgency
Region   rmi://192.168.0.100:6020/MyRegion
Textual UI   True
Trace Errors   True
Trace Warnings   True
Log   True
Logfile   AgencyLog
Security   True

Grasshopper uses private key cryptography for SSL handshake. Hence keys are to be generated. Use Keytool provided by Java to generate keys.

An example of key generation using keytool is geiven below:

```
C:\Documents and Settings\Jaya>keytool -genkey -alias agency1
Enter keystore password:  changeit
What is your first and last name?
  [Unknown]:  jaya gandham
What is the name of your organizational unit?
  [Unknown]:  cs
What is the name of your organization?
  [Unknown]:  iupui
What is the name of your City or Locality?
  [Unknown]:  indy
What is the name of your State or Province?
  [Unknown]:  in
What is the two-letter country code for this unit?
  [Unknown]:  us
Is CN=jaya gandham, OU=cs, O=iupui, L=indy, ST=in, C=us correct?
  [no]:  y

Enter key password for <agency1>
     (RETURN if same as keystore password):  agency1

C:\Documents and Settings\Jaya>keytool -list
Enter keystore password:  changeit

Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry

agency1, May 27, 2004, keyEntry,
```

Certificate fingerprint (MD5): 5D:FD:A3:E9:89:AB:7C:02:6A:CF:31:BF:FD:D7:EB:67

C:\Documents and Settings\Jaya>keytool -export -alias agency1 -file agency1.cer
Enter keystore password: changeit
Certificate stored in file <agency1.cer>

If the Security is set to True, look at the Objects sub-window of the Edit Profile window… click on the System option …it shows two sub-options...Security and Webhopper. Click on Security option…look at the properties window... It shows properties for security…
set Sign Alias to Agency1.

Then, select Sign Storage option under Security…look at the properties window… set the following properties…
Password   changeit
Provider   SUN
Type   jks

Click the Start buntton of the Edit Profile window…
The wizard prompts for Private Key Password for Alias 'agency1'
Enter agency1 and click OK

Then it prompts for keystore Password(client)
Click Cancel

Then it prompts for keystore Password(server)
Click Cancel

The system starts the security service for secure transfer of agents.

The following output in the MS-Dos window of the Agency shows that the security service is successfully started…

C:\Documents and Settings\Jaya>grasshopper
Note: JAVA_HOME is undefined - JSP in Webhopper will not work.

Preparing Wizard [........]
Grasshopper Agent Platform V2.2.4 (C) 1998-2003 by IKV++

18:31:26:828 i AgentSystem: Checking system ...
18:31:26:828 i AgentSystem: Checking '192.168.0.103' (forced) ...
18:31:26:890 i AgentSystem: Hostname is 'Aryan'
  (IP-ADDRESS: 192.168.0.103)
18:31:26:890 i AgentSystem: Initializing security context ...
18:31:26:890 i SecurityContext: Init provider JSSE ...
18:31:27:015 i SecurityContext: Init provider IAIK ...
18:31:27:015 w SecurityContext: Failed, provider is not available!
18:31:27:015 i SecurityContext: Checking available providers ...
18:31:27:015 i SecurityContext: Number of providers: 5
  (SUN 1.42)
  (SunJSSE 1.42)
  (SunRsaSign 1.42)

```
 (SunJCE 1.42)
 (SunJGSS 1.0)
18:31:27:015 i SecurityStorageImpl: Opening keystore 'sign'...
 (LOCATION: file:/C:/Documents and Settings/Jaya/.keystore)
18:31:27:031 i SecurityStorageImpl: Keystore opened
 (PROVIDER: SUN version 1.42)
 (TYPE: jks)
 (ENTRIES: 1)
18:31:27:031 i SecurityStorage: Parsing keystore entries ...
18:31:27:031 i SecurityStorage: Found keypair for alias 'agency1'
18:31:27:031 i SecurityStorage: Password required ...
18:32:36:171 e SecurityStorage: Unrecoverable error: Cannot recover key
18:32:36:171 i SecurityStorage: Opening root CA keystore ...
 (LOCATION: file:/C:/Program Files/Java/j2re1.4.2_05/lib/security/cacerts)
18:32:36:312 i SecurityStorage: Keystore 'ca' opened
 (ENTRIES: 25)
18:32:36:312 i SecurityStorage: Merging ...
18:32:36:312 i SecurityStorage: Found trusted certificate ...
 (ALIAS: equifaxsecureebusinessca1)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: verisignclass4ca)
18:32:36:328 w SecurityStorage: Certificate has expired. Ignoring.
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: entrustglobalclientca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: gtecybertrustglobalca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: entrustgsslca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: verisignclass1ca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: thawtepersonalbasicca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: entrustsslca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: thawtepersonalfreemailca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: verisignclass3ca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: gtecybertrustca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: thawteserverca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: thawtepersonalpremiumca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: equifaxsecureca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: thawtepremiumserverca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
 (ALIAS: entrust2048ca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
```

```
   (ALIAS: verisignserverca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
   (ALIAS: entrustclientca)
18:32:36:328 i SecurityStorage: Found trusted certificate ...
   (ALIAS: baltimorecybertrustca)
18:32:36:343 i SecurityStorage: Found trusted certificate ...
   (ALIAS: geotrustglobalca)
18:32:36:343 i SecurityStorage: Found trusted certificate ...
   (ALIAS: gtecybertrust5ca)
18:32:36:343 i SecurityStorage: Found trusted certificate ...
   (ALIAS: equifaxsecureglobalebusinessca1)
18:32:36:343 i SecurityStorage: Found trusted certificate ...
   (ALIAS: baltimorecodesigningca)
18:32:36:343 i SecurityStorage: Found trusted certificate ...
   (ALIAS: equifaxsecureebusinessca2)
18:32:36:343 i SecurityStorage: Found trusted certificate ...
   (ALIAS: verisignclass2ca)
18:32:36:343 w SecurityContext: Alias 'agency1' not found: signing not available


18:32:36:343 i SecurityContext: Initializing SSL ...
18:32:36:343 i SecurityContext: Trying IAIK ...
18:32:36:343 w SecurityContext: Failed - IAIK not found!
18:32:36:343 i SecurityContext: Trying JSSE ...
18:32:36:359 i SecurityStorageImpl: Opening keystore 'client'...
   (LOCATION: file:/C:/Documents and Settings/Jaya/.grasshopper/security/clientce
rts)
18:32:36:359 i SecurityStorageImpl: Password required ...
18:33:17:609 i SecurityStorageImpl: Loading canceled
18:33:17:609 i SecurityStorageImpl: Opening keystore 'server'...
   (LOCATION: file:/C:/Documents and Settings/Jaya/.grasshopper/security/serverce
rts)
18:33:17:609 i SecurityStorageImpl: Password required ...
18:33:37:437 i SecurityStorageImpl: Loading canceled
18:33:38:781 i SecurityContext: SSL-initialization ok.
18:33:39:750 i AgentSystem: Initializing GUID factory ...
18:33:39:812 i AgentSystem: Initializing ORB ...
   (ARGUMENTS: <none>)
18:33:39:937 i AgentSystem: Initializing communication service ...
18:33:40:218 i DirectoryService: Initializing ...
   (CONTEXT TYPE: Region Registration)
   (PROVIDER: rmi://192.168.0.103:6020)
   (ENTRY DN: MyRegion)
   (USERNAME: <none>)
   (PASSWORD: <none>)
18:33:44:296 i DirectoryService: Initialized.
   (de.ikv.grasshopper.agency.spi.RegionDirCtx@ca6cea)
18:33:44:312 i AgentSystem: Initializing registration service ...
18:33:44:328 i AgentSystem: Start default server ...
   (HOST: 192.168.0.103)
   (PROTOCOL: rmi)
   (PORT: 6000)
```

18:33:44:531 i AgentSystem: Initializing core services.
18:33:44:546 i AgentSystem: Initializing thread service ...
18:33:44:562 i AgentSystem: Initializing listener service ...
18:33:44:578 i AgentSystem: Initializing externalization service ...
18:33:44:578 i AgentSystem: Register agent system ...
18:33:44:578 i RegistrationService: requesting ticket ...
18:33:44:625 i RegistrationService: got Ticket
18:33:44:625 i AgentSystem: Creating default place ...
18:33:44:656 i SecurityManager: Null security manager set...
18:33:44:671 i AgentSystem: Shutdown hook added.
18:33:44:687 i Grasshopper: Loading builtin TUI ...
18:33:44:734 i ListenerService: Listener loaded.
  (CLASS: de.ikv.grasshopper.agency.TextConsole)
18:33:44:734 i Grasshopper: Loading builtin GUI ...
18:33:45:375 i Explorer: Loading desktop ...
  (FILE: C:\Documents and Settings\Jaya\.grasshopper\desktop-MyAgency.ini)
18:33:45:484 i ListenerService: Listener loaded.
  (CLASS: de.ikv.grasshopper.app.explorer.Explorer)
18:33:45:500 i AgentSystem: Event handling started

Agency Text Console (C) 1999 by IKV++
_____
Type 'help [command]' for more information

• DomainSecurityManager(DSM)

java -Djava.security.policy=server.policy DomainSecurityManager <ip address of machine at
which a DSM is to be started> <port number at which a DSM must be contacted>

Example:
rmiregistry 3050

java -Djava.security.policy=server.policy DomainSecurityManager 192.168.0.100 3050

• ExternalCommunicationServer

java -Djava.security.policy=server.policy ExtComServCreator <ipaddress of machine at which an
ExtComServCreator is to be started> < port number at which an ExtComServCreator must be
started > < ip address of machine at which a DSM is running > < port number at which a DSM must
be contacted >

Example:
rmiregistry 3001

java -Djava.security.policy=server.policy ExtComServCreator 192.168.0.100 3001 192.168.0.100
3050

NOTE: ExternalCommunicationService provided by Grasshopper uses port number 3000 in this
application. Therfore, don't use that port number to run any other rmi programs.

- ActiveRegistry

java -Djava.security.policy=server.policy ActiveRegistry <ip address of machine at which an AR is to be started> <port number at which an AR must be started>
<port number of rmiregistry where components must be registered with an AR> < ip address of machine at which a DSM is running > <port number at which a DSM must be contacted> <AR domain> <AR username> <AR password>

Example:
rmiregistry 4000

java -Djava.security.policy=server.policy ActiveRegistry 192.168.0.100 4000 9000 192.168.0.100 3050 Banking Reg1 Reg1

- Headhunter

java -Djava.security.policy=server.policy Headhunter <ip address of machine at which a HH is to be started> <port number at which a HH must be started>
< ip address of machine at which a DSM is running > <port number at which a DSM must be contacted> <ip address of the machine where Region Registry is initialized> <HH domain> <HH username> <HH password>

Example:
rmiregistry 5001

java -Djava.security.policy=server.policy Headhunter 192.168.0.100 5001 192.168.0.100 3050 192.168.0.100 Banking Headhunter1 Headhunter1

- QueryManager

java -Djava.security.policy=server.policy QueryManager <ip address of machine at which a QR is to be started> <port number at which a QR must be started>
ip address of machine at which a DSM is running > <port number at which a DSM must be contacted> <ip address of the machine where Region Registry is initialized>

Example:
rmiregistry 5050

java -Djava.security.policy=server.policy QueryManager 192.168.0.100 5050 192.168.0.100 3050 192.168.0.100

- URDS_Proxy

java -Djava.security.policy=server.policy URDS_Proxy <ipaddress of machine at which a URDS_Proxy is to be started> <port number at which a URDS_Proxy must be started> < ip address of machine at which a QM is running > < port number at which a QM must be contacted > <number of iterations>

Example:

rmiregistry 9009

java -Djava.security.policy=server.policy URDS_Proxy 192.168.0.100 9009 192.168.0.100 5050 2

• Running components

NOTE: All the class files required to run components are under the directory
C:\murds\CompRep\classes. Therefore, always run components under the directory
C:\murds\CompRep\classes.

java -Djava.library.path=. <ComponentName>
 -s <IPaddress of the machine where Active Registry is running:port number where components can
register with an AR/ComponentName>
 -u <codebase of a component's XML file>

Example:
NOTE: Java versions below 1.4.2 accept URL of an XML file as
file://c:/murds/CompRep/bank_xml/account_database_spec.xml whereas Java versions from 1.4.2
accept URL of an XML file as file:\C:\murds\CompRep\bank_xml\account_database_spec.xml.
Therefore, choose appropriate URL format to start a component.

java -Djava.library.path=. AccountDatabase -s 192.168.0.100:9000/AccountDatabase -u
file://c:/murds/CompRep/bank_xml/account_database_spec.xml

java -Djava.library.path=. AccountDatabase -s 192.168.0.100:9000/AccountDatabase -u
file:\c:\murds\CompRep\bank_xml\account_database_spec.xml

java -Djava.library.path=. ATM -s 192.168.0.102:9000/ATM -u
file:\c:\murds\comprep\bank_xml\atm_spec.xml

java -Djava.library.path=. CashierTerminal -s 192.168.0.106:9000/CashierTerminal -u
file:\c:\murds\comprep\bank_xml\cashier_terminal_spec.xml

java -Djava.library.path=. CashierValidationServer -s 192.168.0.103:9000/CashierValidationServer
-u file:\c:\murds\comprep\bank_xml\cashier_validation_server_spec.xml